Components as Coalgebras

Luís Manuel Dias Coelho Soares Barbosa

Departamento de Informática
Escola de Engenharia
Universidade do Minho
2001

Tese de doutoramento em Informática, área de Fundamentos da Computação, aprovada por unanimidade em provas públicas realizadas na Universidade do Minho

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA E DA TECNOLOGIA

This thesis has been partially supported by the LogComp *and* Karma *projects, under, respectively, contracts* PRAXIS XXI 2/2.1/TIT/1658/95 *and* P0011/IC-PME/II/S - Karma. *Its publication was generously supported by* FCT*, the Portuguese Foundation for Science and Technology, and the* Algoritmi *Research Center.*

Abstract

In the tradition of mathematical modelling in physics and chemistry, constructive formal specification methods are based on the notion of a *software model*, understood as a state-based abstract machine which persists and evolves in time, according to a behavioural model capturing, for example, *partiality* or (different degrees of) *nondeterminism*. This can be identified with the more prosaic notion of a *software component* advocated by the software industry as 'building block' of large, often distributed, systems. Such a component typically encapsulates a number of services through a public interface which provides a limited access to a private state space, paying tribute to the nowadays widespread object-oriented programming principles.

The tradition of communicating systems formal design, by contrast, has developed the notion of a *process* as an abstraction of the behavioural patterns of a computing system, deliberately ignoring the data and state aspects of software systems.

Both *processes* and *components* are among the broad group of computing phenomena which are hardly definable (or simply not definable) algebraically, *i.e.*, in terms of a complete set of constructors. Their semantics is essentially *observational*, in the sense that all that can be traced of their evolution is their *interaction* with the environment. Therefore, *coalgebras*, whose theory has recently witnessed remarkable developments, appear as a suitable modelling tool.

The basic observation of category theory that *universal* constructions always *come in pairs*, has motivated research on the duality between algebras and coalgebras, which provides a bridge between models of *static* (constructive, data-oriented) and *dynamical* (observational, behaviour-oriented) systems. At the *programming* level, the intuitive symmetry between *data* and *behaviour* provides evidence of such a duality, in its canonical *initial-final* specialisation.

This line of thought entails both definitional and proof principles, *i.e.*, a basis for the development of program calculi directly based on (actually driven by) type specifications. Moreover, such properties can be expressed in terms of *generic* programming *combinators* which are used, not only to calculate programs, but also to program with.

Framed in this context, this thesis addresses the following main themes:

- The investigation of a semantic model for (state-based) *software components*. These are regarded as concrete coalgebras for some Set endofunctors, with specified initial conditions, and organise themselves in a bicategorical setting. The model is able to capture both behavioural issues, which are usually left implicit in state-based specification methods, and interaction through structured data, which is usually a minor concern on process calculi. Two basic cases are considered entailing, respectively, a 'functional' and an 'object-oriented' shape for components. Both cases are *parametrized* by a model of behaviour, introduced as a strong (usually commutative) monad.
- The development of corresponding component *calculi*, also parametric on the behaviour model, which adds to the genericity of the approach.
- The study of *processes* and the 'reconstruction' of classical (CCS-like) process calculi on top of their representation as inhabitants of (the carriers of) final coalgebras, in an essentially *pointfree*, calculational style.
- An overall concern for *genericity*, in the sense that models and calculi for both components and processes are *parametric* on the behaviour model and the interaction discipline, respectively.
- The *animation* of both processes and components in CHARITY, a functional programming language entirely based on inductive and coinductive categorical data types. In particular this leads to the development of a process calculi interpreter parametric on the interaction discipline.

Preface

Conforme a vida que se tem o verso vem
Alexandre O'Neill, 1965.

Every thesis has a context and this one is no exception. As an undergraduate, I was in the small group of students who attended the first formal methods class run by José Nuno Oliveira at Minho. His lively lectures on programming as a *true* engineering discipline, led a definitive mark in all of us. Shortly afterwards we found ourselves struggling to apply such methods in industrial projects. I remember myself encoding systems' invariants back into commercial languages to filter corrupted registers in production databases. Since then, I have been involved with model-oriented specification methods. And eventually I became interested in ways of expressing models behaviour and defining composition mechanisms able to take into account the dynamical and architectural issues present in most systems. My first academic dissertation was precisely an attempt to combine VDM with behavioural specifications given as process algebra expressions or Petri nets.

When José Nuno Oliveira became my supervisor his main research was oriented toward the development of a transformational calculus for data refinement, named SETS after *Specification in Set*. Would a similar calculational style scale up to a setting where software specifications (or *components*, as we used to call them) included some behavioural features? What would then be the composition patterns with respect to which a refinement relation could be proved monotonic? I would like to regard this thesis as a first step in that direction.

In the summer of 1997, however, most of my understanding of what I was trying to achieve was confronted by the contact with a few papers on coalgebra theory by

Bart Jacobs and Jan Rutten. The simple notion of a coalgebra came almost as a revelation in the sense that several constructions and results I had been studying in transition systems and process calculi appeared to have found their right conceptual place and achieved a greater level of genericity. This 'mid-summer' readings re-shaped all my work and led to the present outcome.

Later, searching for simple ways to animate my constructions, I visited the CHARITY web-page. The result was almost a *coup de foudre*: the language provided explicit support for coinductive types, enforced a discipline for their use, had a sober syntax and an unusually precise semantics. Bits of CHARITY code are, therefore, spread along this thesis. It was my intention not only to give a more concrete flavour to the text but also to suggest an approach to prototype dynamical systems and their calculi as a vehicle to assess alternative design decisions. Having developed such bits of code almost with no contact with the CHARITY group, I had the chance of meeting Robin Cockett during CMCS'01. I am most grateful for his sharp comments and the interest shown in my prototypes.

By that time I was also involved in the design of CAMILA, a functional prototyping tool for the SETS calculus. It is a pleasure to acknowledge the close collaboration in this project with José João Almeida, my former roommate and a friend of all times. The CAMILA seminar I have been running every year, since 1990, at the University of Bristol as well as the projects I had the chance of supervising there, provided an important feedback on both the tool and the method. I believe to have learnt a lot along this process even if, in a sense, it may be partially responsible for the retardation in finishing the thesis.

During all this time José Nuno Oliveira was not only my formal supervisor, but also a fruitful source of ideas and questions. I wish to thank him for the confidence he has deposited on me, his enthusiasm, rigour and active interest in my research. I owe him a lot.

I also wish to thank the *Departamento de Informática*, at Minho, for the smooth working environment I had the chance to benefit from. My gratitude goes to José Valença and Francisco Moura, respectively its former and present directors. I am also pleased to mention my colleagues of the former Computer Science group for their support and the quality of their friendship. In particular, José João Almeida for so many technical and non technical morning coffees, and José Barros, who was most helpful in alleviating my teaching loads in critical periods. And, of course, Pedro

Henriques and his ability to blend, inside and outside academia, 'safety' and 'liveness', with 'strong fairness' and generous 'availability'. Last but not least, a word for the GEF people and our sort of 'Wednesday afternoon club' where some of us struggled with Mac Lane's book, Stone dualities and similar stuff.

Parts of the work reported in this thesis benefitted from comments of several people. In particular, I wish to thank Jeremy Gibbons, who gave me the opportunity of presenting it during the ACMMPC Summer School, in Oxford. Let me also mention Alberto Pardo, Fernando Luís Neves, Gilles Barthe, João Saraiva, José Barros, José Carlos Bacelar, José João Almeida, José Valença and Tarmo Uustalu. To all of them, in different occasions, I owe the discussion of an intuition, a reading suggestion, a comment on a detail or a challenging question.

My gratitude extends, of course, to the members of the Assessment Committee, in particular, my external examiners, J. J. M. M. Rutten (CWI, Amsterdam) and Luís F. Monteiro (UNL, Lisbon), for carefully reviewing the thesis.

Finally, I would like to thank my family and friends, who often wondered whether this thesis would ever be finished, for their support and encouragement. To my parents for everything they have made possible. They have brought me up stimulating a variety of interests and the deep conviction that a question mark is far more interesting than a full stop. But, at the same time, the awareness that what really matters is probably far behind what one may achieve in formal education or professional life. Since childhood, Aunt Lena has always been around. She taught me words and numbers, the ways of combining them, and combining the composites yet again. This is basically what I am still doing.

Above all, I was lucky to have met Lena, her love and the moving light of her eyes. For a long time she has tirelessly put up with many of the most stressful *things that have to be done*. The rest cannot be put into words. This thesis is dedicated to our children

Rui, *Nuno* and *Catarina*

a lively 'diagram' always ready to 'commute' ... in 'non unique' ways.

Contents

CHAPTER 1

Introduction

Summary

This chapter provides the context for and states the problem addressed in this thesis: what software components 'are' and how associated calculi can be designed on top of coalgebraic component models. The thesis structure and contributions are outlined, providing a 'roadmap' for the chapters to follow.

1. The Problem and Its Context

1. COMPONENTS. The expression *software component*, like many others in programming engineering, is so semantically overloaded that referring to it is often a risk. Therefore, we would like to make it clear, from the outset, what is understood by a *software component* in this thesis: the *specification* of a *state-based* module, eventually acting as a 'building block' of larger, often concurrent, systems. Again that qualification *concurrent* may convey several different meanings, *e.g.*, independent evolution, cooperation to achieve a common goal, competition for a shared resource. Such a component should encapsulate a number of services through a public *interface* which provides limited access to its internal state space. Furthermore, it *persists* and *evolves* in time.

In summary, our model for components has to deal with, at least,

- the presence of an *internal state space*, acting as the component 'memory',
- the possibility of *interaction* with other components during its overall computation,
- the existence of input and output *observation universes* to ensure the flow of data.

2. MOTIVATION. The notion of a software component studied in this thesis arises from a widespread paradigm for formally approaching systems' design: the so-called *model oriented* specification method, of which VDM [Jon86] and Z [Spi92] are well-known representatives. In such methods, an explicit *model* of the system being specified, rather than an axiomatic *theory* capturing its requirements, is defined. Usually a distinguished sort plays the role of a state space. State is often taken as a 'black box' accessible only via specified operations, which justifies the qualification of *state-based* typically given to such methods. Data and functionality are explicitly defined, while the temporal ordering of operation calls and, in general, most behavioural issues, are left implicit. Nevertheless, most aspects mentioned in §1 about the informal characterisation of components may be found in what is called a *model* in the VDM meta-language, a *schema* in Z, a *machine* in B [Abr96] or a *component* in CAMILA [ABNO97].

In broad terms, the research reported in this thesis finds its motivation in the use of these methods in software projects and, in particular, in the development of the CAMILA experimental platform for model-oriented software development [ABNO97]. This encompasses an (executable) set-theoretic notation and an inequational *calculus* of data structures [Oli90, Oli92b].

However, when using model-oriented specification methods it is often difficult to identify and reason about the behavioural patterns underlying a specification and their relative (in)dependence with respect to data and functionality descriptions. Often the only behavioural information that can be extracted from, say, a VDM specification amounts to the local constraints recorded in the operations' pre-conditions. This is, however, implicit and does not carry enough information neither on the systems architecture nor on the intended usage.

On the other hand, such methods lack appropriate structuring mechanisms to build, in a compositional way, specifications of open, distributed and/or concurrent software architectures. A diagrammatic notation, reminiscent from circuit design, to express the composition of state-based modules was proposed in [Oli91], as a base for the *reuse* of CAMILA specifications [Oli93]. Even though the semantics of such a notation has remained informal, basic intuitions from this work acted as an initial motivation for this thesis.

3. OBJECTIVES. Within the research context mentioned in §2, this thesis intends to address the following topics:

- The definition of a semantic model for *software component* specification entailing a suitable characterisation of *observational* equivalence and explicitly parametrized by a notion of *behaviour*. For example, the behaviour model of some components may cater for the possibility of termination even though, in concurrent systems, the existence of '*terminal states might denote disaster rather than result*' [Rei88]. Other components may require the explicit specification of (eventually different degrees of) nondeterminism. The specification of behaviour seems to be orthogonal, to the description of system's functionality, but is often overlooked in the above mentioned state-based formalisms.

- The definition of a set of *component combinators*, able to take into account the dynamical and architectural issues present in most modern computational systems. Of course, different assumptions on the components' model (either at the *interface* or *operational* levels) may entail corresponding specialisations of the combinator set. In any case, however, their definition has to be parametrized by the component's behaviour model.
- The development of associated *component calculi* to reason about and transform component based designs. The *calculational style*, often referred to, in the context of functional programming, as the *Bird-Meertens formalism* (see discussion in §14), is a cornerstone in the approach to software engineering at Minho (see [Oli98] for a tutorial presentation). We would like to scale it up to the level of components and their behaviour.

4. CONTEXT. The informal characterisation given so far suggests that a component specification has to deal with both *data* (static) and *behavioural* (dynamic) structures. A third fundamental element is *interaction*. In a sense, the history of (theoretical) Computer Science records the efforts in understanding such basic ingredients of any computational system and their interplay[1].

The following section proposes a brief visit to some 'milestones' in the literature in order to settle the background of this thesis. We shall stress, in particular, the general categorical approach to inductive and coinductive types and the underlying duality between *algebraic* and *coalgebraic* structures in computation. This duality is the formal basis of the 'data' *vs* 'behaviour' symmetry mentioned above. As expressed by its own title, such is the framework adopted in the thesis to discuss software components.

Finally, in section 3, the thesis structure and contributions are outlined, as a sort of a 'roadmap' to the chapters to follow.

[1] A word of advice: not only the interplay but also some form of *interchangeablity* between data and behaviour are known from programming practice. Recall, for example, data-oriented *vs* algorithmic-oriented solutions to the parsing problem [Knu65] and how the former seems to encode the latter. Or the representation of data structures by CCS [Mil89] or CSP [Hoa85] processes.

2. Background

5. DATA. David Parnas [Par72] is usually credited as a pioneer in recognising that operations (algorithms) should be associated with data representations, just as functions are associated to sets in Universal Algebra. From then on the theory of *abstract data types*, starting with the work of the ADJ group [GTWW77, GTW78], emerged as a fundamental cornerstone in programming theory. It provides an abstract description of *data structures* in semantics, regarded as algebraic structures encapsulating data sorts and operations upon them. A major concern of this line of research was to ensure representation independence of data as well as to facilitate formal manipulation.

Reasoning about data structure specification and implementation resorts to the techniques and methodology of Universal Algebra — a well established branch of Mathematics, which, at least symbolically, dates back to the publication of Birkhoff 'variety theorem' [Bir35] in 1935.

The success of the approach is greatly acknowledged both in the semantics and pragmatics of software specification (see, for example, [EM85] as a basic reference at a textbook level and [Wir90] for a tutorial). In particular, it has lead to the development of specification languages and systems (such as OBJ [GWM$^+$96], CIP-L [BBB$^+$85], CASL [CoF95] or LARCH [GH93], among many others) and to a broader understanding of the nature of algorithms, their classification and transformation. Extensions of this approach are referred to in §9-11, below.

6. PROCESSES. On the other hand, looking at behavioural patterns in an algebraic way has been the overall research program on *process algebras* [Mil80, Hoa85, Hen88, Mil89, BW90b, Mil99] for the last two decades. This has been carried to an extent that discards the actual observed data. Actions are just symbols in a formal language. Such techniques embody a specification style which is in sharp contrast with the state based methods mentioned in §2. In particular, neither the state space, nor in fact any data sort, is explicitly defined, the emphasis being put on the temporal ordering of interactions.

Such approaches have been successful in dealing with highly interactive systems, in particular with those where complex control algorithms have to be expressed and where data plays a comparatively minor role. Such is the case, for example, of the specification of communication protocols, real time control algorithms or control mechanisms in operating systems.

C. A. Petri [Pet62], in the 1960´s, was probably the first to put forward a model for *concurrent* computation, based on network transitions. The theory of networks named after him ('Petri nets', see [Rei85] for a comprehensive tutorial), makes it explicit that a transition in a system may *interact*, or depend upon, another transition occurring in a different system executing alongside.

Since then, there has been a proliferation of both semantic models for concurrency and associated calculi. Reference [WN95] provides a well structured and comprehensive survey. Reference [Sti92] is a tutorial on specification of concurrent processes in *modal* and *temporal* logics, a line of research made popular after A. Pnueli landmark paper [Pnu77]. Associated verification methods are discussed at textbook level in [MP92]. In any case, the diversity of semantic models proposed (emphasising *e.g.*, linear or branching temporal structures, causality or interleaving, synchrony or asynchrony) witnesses both the difficulty of understanding concurrency and the practical relevance of the topic.

Even in the particular area of *process algebras*, such a diversity is overwhelming. It manifests itself on the set of combinators chosen, interaction disciplines adopted, application domains (ranging from the 'general-purpose' mainstream to highly biased calculi), semantic presentations (operational, as in CCS [Mil89], denotational or axiomatic, as in CSP [Hoa85] or ACP [BW90b], respectively) and notions of equivalence entailed. In reference [vG90], for example, as many as 155 different semantics are classified from a modal logic unifying point of view. The absence of a canonical calculus or theory for concurrent behaviour is commented, with a bit of irony, by J. Goguen in [GM00] as follows:

> *The lack of consensus on a suitable set of equations is discouraging, suggesting that these 'laws of concurrency' may not have the same status of 'laws of nature' in physics, despite occasional claims on the contrary.*

Or, as S. Abramsky [Abr94] puts it, '*may be some key ideas are still missing*'.

Reference [Abr94] is the original paper on *interaction categories* (see [AGN94] for a detailed tutorial): an attempt to abstract away from concrete process calculi and single out the core structure which processes collectively possess in the form of a *categorical model* [Mac71]. In an interaction category, objects are process *types* (or specifications), arrows are processes and composition is regarded as interaction. For example, in the interaction category of synchronous systems, types are the so-called 'safety specifications' (*i.e.*, an action alphabet *Act* and a set of traces, *i.e.*, a non empty prefix closed subset of Act^*, representing the possible evolutions), processes are synchronisation trees satisfying the 'safety specifications' on types and interaction is understood, as in [Mil89], as a combination of parallel composition with restriction.

Note that, in general, there is no implicit *direction* in the arrows: the split of a process observation universe in merely conventional. A generalized *monoidal* category[2] is proposed as the appropriate structure to express the *spatial*, or 'architectural',

[2] More precisely, a *linear* category [See89]. Technically, a symmetric monoidal category, closed (in the sense that the tensor $\otimes$ has a right adjoint $\multimap$ which provides an internalisation of the hom sets, just as it happens with cartesian product and exponentials in the category Set of sets and functions), with a

organisation of processes. The *temporal* dimension, usually given by the *prefix* combinator in process calculi, is captured by 'delay monads' on top of the linear structure. As expected, having axiomatized such a basic, typed framework for processes, a lot of further structure can be 'found' (*i.e.*, determined up to isomorphism) and this categorical pay-off is both the motivation and the strength of the approach. Suitable instantiations of such a framework for different interaction disciplines give rise to different interaction categories, capturing *e.g.* asynchrony and even the familiar landscape of sequential programming and interaction by function calling [Gay95].

Over the last decade, another major contribution in this area has been the development of *mobile* calculi, after Milner's work on the π-*calculus* [MPW92]. The basic idea is a notion of *naming* (a generalisation of 'reference'), name passing (generalising 'value' or 'message passing') via named channels and name (channel) dynamic creation.

Both mobile calculi and interaction categories, having almost nothing in common in their genesis, method or theory, take *interaction* as the basic, pervasive, notion in computing, generalising previous work on process calculi. Both aim to provide semantic foundations to programming languages able to combine interaction and concurrency with types, high-order constructs and polymorphism.

7. SYSTEMS. Abramsky's *interaction categories*, mentioned in the last paragraph, are paradigmatic of a class of specification approaches which resort to category theory to build *typed* frameworks for processes — see also, among many others, [KSW97a, KSW97b, Spo97, CPW98, Sel99]. Most of these approaches abstract from the internal state of a system and endow its interfaces with extra structure (*e.g.* a predicate on the allowed traces) in order to capture particular interaction paradigms.

Departing from this trend, one finds the work of R. Walters and his collaborators who consider both an explicit description of systems' internal dynamics and their behavioural semantics. Processes, in this sense, are dynamical systems, with an internal state and input-output interfaces carrying just enough information to assure the flow of data. The ability to talk both in terms of *behaviours* (which is the only concern in most models of concurrency) and *machines* suggests a *bicategorical* framework: Walters' landmark paper [KSW97a] is suggestively entitled '*Bicategories of Processes*'.

Such bicategories are generated from an underlying symmetric monoidal category, leading to an algebra mimicking the basic 'assembly modes' in conventional Engineering: *series* (composition), *parallel* (tensor product) and *feedback* (usually

dualizing object —*i.e.*, an object $\perp$ such that any object X is isomorphic to $(X \multimap \perp) \multimap \perp$ —and finite products. The category Rel of sets and relations is a prime example.

taken as a *trace*, in the sense of [JSV96]). Different notions of behaviour are captured in a functorial way involving appropriate semantic categories (*e.g.*, relations on streams).

The overall aim is to develop *compositional* models of dynamical systems using monoidal bicategories equipped with some form of feedback operations. Two cases illustrating the kind of system addressed in this approach are discussed in [KSW97a] (and further detailed in P. Katis thesis [Kat96]) : *circuits* and *flowcharts* based on the monoidal structure imposed on the category of sets by, respectively, Cartesian product and disjoint union. This line of thought is documented in a series of papers of which [KSW00] provides a comprehensive overview. A particularly successful application can be found in the development of an alternative approach to (asynchronous) circuit theory — see H. Weld thesis [Wel98] and, more recently, [KSWW01]. This last reference introduces a family of traced monoidal categories of binary circuits (and corresponding behaviour notions). A number of circuit classes are discussed paving the way to a formal taxonomy, which by itself provides evidence of the maturity of the approach.

The semantic model for software components proposed in this thesis (namely, in chapter 5) is strongly influenced by this work. Therefore, we shall come back to it later, when discussing the thesis contributions (in chapter 7).

8. HYBRID FRAMEWORKS. There are several accounts in the literature of combining process calculi with some sort of mechanism for the specification of the data handled by processes. Perhaps the best known example is LOTOS [BB87, ISO88], which combines an axiomatic, algebraic specification language (ACT ONE [EFH83]) with CCS. LOTOS is an official ISO specification language for open distributed systems. Reference [GP95] provides another example in the same line of research. Such hybridizations are not restricted to process algebras: they also appear, for example, associated to temporal logic specifications [SFSE89, CR97] and Petri nets. In algebraic Petri nets [Rei91], for example, elements of the 'token' set, whose distribution along the net represents its global state, are regarded as an algebraic specification of a data structure.

Combinations of behavioural formalisms with state-based specifications have also been studied. An important reference is [BD99] which summarises a long research collaboration between the authors around the semantic integration of Z and CSP. The literature on this sort of hybrid approaches is extensive and eventually also considers the incorporation of object-orientation (§10) features, see *e.g.*, [DRS95, Fic97]. Such combinations are often placed at the methodological level, with a pragmatic intention, and less often at the level of semantics. Some mere notational overlaps, still quite common, are not worthwhile mentioning.

Often a paradigm may put under new light an old problem of a different one, and this is probably the right justification for comparative research. An interesting example is C. Jones proposal [Jon96] of a design notation extending VDM with some object primitives to address the control of *interference* in concurrent systems, a problem which, as he himself recognises in [Jon83], may invalidate conventional pre/post-condition reasoning.

9. Others have thought that perhaps a deeper interplay between algebraic specifications and behavioural descriptions consists in either enriching an algebraic data specification with sorts whose inhabitants correspond to processes (or states of transition systems), or in allowing the specification of a data type itself to evolve in time. A well known example of the former is J. Meseguer's rewriting logic [Mes92] in which actions are regarded as inference steps in a logic which is essentially equational logic without symmetry. Rewriting logic is animated in MAUDE (see [Mes00] for a recent, detailed overview), whose dynamics is based on the concurrent transformation of a 'soup' of objects and messages.

On the other hand, the latter approach associates a different algebraic specification to each configuration of the system: a state transformation becomes a transition from one algebra to another. Such an approach, first proposed in the so-called 'reflexive semantics' for object orientation [GM87], is particularly biased toward this paradigm (§10), rather than to concurrency semantics. The underlying theory is implemented in FOOPS [GM87, RS92] which, just as MAUDE, is an extension of OBJ (§5).

A similar direction is taken by 'evolving algebras' [Gur93] (also known as 'abstract state machines'), in which systems are described as labelled transition systems whose states are algebras.

10. OBJECTS. *Object-orientation* [GR83, Mey88] has emerged in the last decades as a popular programming paradigm, whose origins can be traced back to the SIMULA proposal [DMN68]. Such an informal concept of object as an entity whose 'memory' and 'identity' persists over time, offering a controlled access through an interface of *attributes* and *methods*, was intuitive and immediately attractive to programmers. However, programming languages that support object-orientation pose difficult and subtle semantic problems and the paradigm still lacks a commonly recognised formal foundation (see references [SFSE89, ESS90, HP95, Pra95, AC96], as well as the 'algebraic' approaches mentioned in next paragraph). Most research has focused on type systems and models for inheritance, but also, more recently, on local state and interaction issues. [GK96], and the references therein, give an updated account of problems and achievements.

Semantically, an *object* can not be regarded as a process in the sense of §6 above, as static aspects are also involved. On the other hand, it cannot be simply modeled by an algebraic structure as it relies on an internal state, typically regarded as a 'black box', and exhibits non functional behaviour.

From a practical point of view, it has been recognised that object-oriented languages, successful as they are for implementing and packaging software components, offer too limited mechanisms for their interconnection. As [ND95] points out, a possible explanation is that *'object-oriented source code exposes class hierarchies, but not object interactions'*.

11. BEHAVIOURAL SATISFACTION. Research on foundations of object-oriented languages and, in particular, on what a suitable notion of a *type* for objects could be, has stressed some symmetries with the theory of *abstract data types* (§5). The key idea underlying such extensions is that of *behavioural satisfaction*, after H. Reichel's [Rei81] seminal work on possible unifications of *initial* and *final* semantics (§13). It conveys the intuition that the satisfaction of a specification by a computational system is usually not strict but *observational*, in the sense that it appears to be satisfied under any experiment carried out on the system. Therefore, behavioural equivalence stands for equality under all experiments or observation contexts. An early reference on *observational semantics* for algebraic specifications is [GGM76], but see also [ST85] for an overview. Another OBJ-like language, CAFEOBJ [DF98], was the first to automate behavioural satisfaction.

A important development, leading to a recently active research area, is the 'hidden-sort algebra' approach. Hidden-sort algebra embodies a fundamental distinction between data that is used to model values (*e.g.*, in object attributes) and data used for internal states, which can only be observed in an indirect way. Therefore data sorts are divided into visible and hidden, the satisfaction of equations involving the latter being proved only up to the observable output — *i.e.*, the result of experiments, which technically correspond to 'contexts', *i.e.* terms typed in visible sorts. The original papers on this subject are [Gog91] and [GD94] and a comprehensive introduction can be found in [GM00]. The integration of static and dynamic aspects of the object paradigm in this setting is studied in [Veg97]. A logic for reasoning on hidden specifications and the implementation of a support system — BOBJ — appeared recently in G. Rosu's thesis [Ros00].

12. A BASIC DUALITY. After this brief incursion around the notions of *data structure*, *process* and *object*, time has come to frame them in a more general setting. The main point to emphasise is that the intuitive symmetry between *data* and *behaviour* that pervades Computer Science, can be understood as a precise mathematical *duality* between *algebraic* and *coalgebraic* structures. Concise notions of what such structures are and their dual nature are revealed under the light of *category theory* [Mac71].

As mentioned above in §6, when reviewing 'interaction categories', in a categorical setting models of computation are represented by categories whose objects and arrows model respectively *types* and *computations*. In fact, the original work on abstract data types referred in §5, was carried out in such a basis. Therefore, data type *signatures*, thought of as their external interfaces, define *type constructors* modeled as endofunctors in the appropriate category. Given one such a functor, say T, a corresponding algebraic structure (*i.e.*, a T-algebra) is simply a map

$$d : \mathsf{T}\ D \longrightarrow D$$

which specifies how values of D are generated using a collection of *constructors*, whose signature is registered in the 'shape' of T. A canonical representative of the envisaged structure arises as a (least) solution, *i.e.*, a fixpoint, of equation $X \cong \mathsf{T}\ X$. It corresponds to the *initial* (term) algebra for this signature. Recall that initiality of the term algebra means there exists a unique homomorphism from it to any other T-algebra. The two sides of this *universal* property — *existence* and *uniqueness* — entail, respectively, the so-called *inductive definition* and *proof* principles (see [GTW78] for an early account).

An advantage of working inside a categorical model is that, once the basic definitions are settled, a lot of structure comes for free. The 'categorical trilogy' — *functoriality*, *naturality* and *universality* — leads to canonical constructions by constraining what should be acceptable as a structural characterisation. Universal constructions, in particular, are paradigmatic: if they exist, they do so in essentially an unique way[3]. Categorical frameworks convey, in fact, what S. Abramsky calls a *'major shift from stipulation to observation of structure'* [Abr94]. Moreover, thinking categorically provides further levels of abstraction, in the sense that underlying mathematical 'universes' need not to be fixed once and for all.

Another 'magic' word in the categorical way of thinking is *duality*, which basically means that universals always *come in pairs*. As discussed below, 'arrow reversal' paves the way to the generalisation of *constructive* (initial, inductive) data structures (like the natural numbers or finite trees) to the dual realm of *observable* (final, coinductive) structures able to model 'infinite' or 'circular' objects or, more generally, to provide semantic universes for *behaviours*.

[3] A popular example, related to polymorphism in programming, is Wadler's *theorems for free* [Wad89].

13. COALGEBRAS. As a matter of fact, there are several phenomena in computing which are hardly definable (or even simply not definable) as algebras, *i.e.*, in terms of a complete set of constructors. Think, again, of processes, transition systems, objects, stream-like structures used in lazy programming languages, 'infinite' or non well-founded objects arising in semantics, and so on. Such 'systems' are inherently dynamic, do possess an observable behaviour, but their internal configurations remain hidden and have therefore to be identified if not distinguishable by observation. Furthermore, the formalisation of the possibly infinite behaviour they may exhibit requires infinite structures, whereas elements of an initial algebra are always finite. *Coalgebraic structures* seem appropriate to deal with such issues.

While data entities in an algebra are built by constructors and considered to be different if differently constructed, coalgebras deal with entities which are observed, or decomposed, by observers (or 'destructors'). Any two internal configurations are identified if they cannot be distinguished by observation. Given an endofunctor T, a T-coalgebra is a map

$$p : U \longrightarrow \mathsf{T}\, U$$

which may be thought of as a transition structure, of *shape* T, on object U, usually referred to as its *carrier* or *state space*. The shape of T describes not only the way the state is (partially) accessed, through *observers*, but also how it evolves, through *actions*. T specifies a signature of actions and observers over a carrier but it omits its constructors. As a consequence equality has to be replaced by *bisimilarity* (*i.e.*, equality with respect to the observation structure provided by T) and *coinduction* replaces induction as a proof principle.

The dual concept to *initial* algebra is that of *final* coalgebra. For a given T, it consists of all possible behaviours up to bisimilarity, in the same sense that an initial algebra collects all terms up to isomorphism. It is also a (greatest) fixpoint of a functor equation and provides a suitable universe for reasoning about behavioural issues. In this context, final coalgebras are called *coinductive* or *left datatypes* in [Hag87b] or [CS92], *codata* and *codatatypes* in [Kie98b, Kie98a], *final systems* in [Rut00] or *object types* in [Jac96b].

The relevance of coalgebraic concepts and tools was first recognised in programming semantics — see, for example, P. Aczel foundational work on 'non well-founded sets' and the semantics of processes [Acz88, Acz93]; H. Reichel characterisation of behavioural satisfaction [Rei81] and J. Rutten, G. Plotkin and D. Turi work on final semantics [RT94, Tur96, TP97]. The methodology of *final* semantics starts from a universe of 'behaviours' built as a final coalgebra for a functor T determined by the *operational* semantics of the language. Then, syntax is casted into a T-coalgebra such that behaviour equivalence is the equivalence induced on terms by the unique arrow to the final 'universe'. The whole process is *semantics-driven* — one looks for an ‘explanation’ of behaviour. Dually, *initial semantics* [GTWW77] is *syntax-driven*: one starts from a term algebra for a functor T and builds another T-algebra of ‘meanings’ (or ‘denotations’) of the syntactic objects. The semantic map is now the unique arrow from the initial algebra. The approach is difficult to apply to the sort of ‘infinite’, ‘operational’ or ‘circular’ entities mentioned above, leading to heavy mathematical structures and contrived encodings. A basic reference on the final semantics approach is [TR98]; detailed applications are given in M. Lenisa’s thesis [Len98]. Aczel’s excellent tutorial [Acz97] provides a motivating introduction.

Although previously known in Universal Algebra, coalgebras began to be seriously considered only after the categorical account of both algebraic and coalgebraic structures of a *type* T (*i.e.*, for an endofunctor T in an arbitrary category) has provided the right generic framework in which several phenomena and theories fit. The systematic study of their theory, essentially along the lines of Universal Algebra, was initiated by J. Rutten in [Rut96] (later published as [Rut00]). This rapidly expanded to a broad research topic, as an increasing number of contributions appeared on both the theory, applications and rephrasing of ‘old’ results in seemingly unrelated areas. Part of this research is documented in the proceedings of the *Coalgebraic Methods in Computer Science* workshop series, starting in 1998. Reference [JR97] provides an introductory tutorial.

14. CATEGORICAL DATA TYPES. The study of such a broad notion of abstract data type, in the *initial algebraic* and *final coalgebraic* trends, dates back to the ADJ group in the 1970’s (and also [Wan79, Rei81]). It was later ‘rediscovered’ and brought back to fashion by T. Hagino landmark thesis [Hag87a]. This research area is now usually referred to as the theory of *categorical data types* (some early contributions include [Hag87b, Wra88, Mal90a]).

Generic combinators, parametric on the functor encoding the type signature, arise as universal arrows. They encode several recursion patterns which depend on the *shape* of the structure which the algorithm consumes, generates or, simply, ‘rests on’ (as a virtual data structure). In particular, *iteration* (respectively, *coiteration*) is modelled by the inductive (coinductive) extension of an algebra (coalgebra), *i.e.*, the unique arrow from (to) the initial (final) algebra (coalgebra). Such constructions become known in the area of *constructive algoritmics* [Mal90b, Fok92b, BM97], as, respectively, *catamorphisms* and *anamorphisms*. A number of specialisations of such basic schemata have been proposed, several of which are covered in M. Fokkinga thesis [Fok92b], with an emphasis on the inductive side (see also [UV99] for a recent development). L. Meertens introduced *paramorphisms* in [Mee92] which correspond to *primitive recursion*. The dual notion of *apomorphism* is due to T. Uustalu and V. Vene [VU97]. Recently, more generic schemes have been studied: they are not only parametric on the functor capturing the type signature, but also on a comonad with a distributive law which encodes the recursive call pattern of a particular recursion scheme for the underlying inductive type (see [UVP01] and, for a similar motivation on the coinductive side, [Bar01b]).

Such a research area, devoted to the development of program calculi directly based on, *i.e.*, actually driven by, type specifications, builds upon both the *genericity* and the *calculational style* [Fok92a] entailed by category theory, the latter arising from the fact that most categorical properties can be formulated as (usually equational) laws. This has had a fundamental impact on algorithm derivation and transformation, mainly on the framework of *functional programming*. In fact, since John McCarthy original papers [McC60, McC63], and, later, Backus FP [Bac78] — a functional language based on combinators related by algebraic laws — functional programming and *program calculi* have developed in an intertwined way. Around the end of the eighties, the so-called *Bird-Meertens formalism* [Bir87, BM87], originally an equational theory of sequences which formed the basis of a calculus for transforming list based programs, had emerged. R. Backhouse [Bac88] published on the basic role of category-theoretic universals in programming and G. Malcolm [Mal90b] made the community aware of the foundational work of Hagino. Since then the area has known a remarkable progress, as witnessed by the vast bibliography published on both the theory and applications (*e.g.*, [MFP91, Mee92, Fok92b, BH93, Jeu93, Aug93, Gib93, BM94, DM94, Fok96, Hoo96, Par98, VU97]; see also [BM97] as a textbook and [BJJM98] for a tutorial introduction).

Furthermore, such generic combinators have been incorporated on real programming languages as *polytypic* functionals, generalising the well-known `map`, `fold` and `unfold` constructs, in functional languages (see, eg, [She93, JC94, JJ97]). There are also programming languages entirely based on categorical data types. CHARITY [CF92] is probably the paradigmatic one. D. Kieburtz ADL [KL95] (see [Kie98b] for applications to reactive programming) should also be mentioned.

In programming, the basic symmetry between initial and final types makes itself more intuitive in languages, like CHARITY, in which all programs are guaranteed to terminate (in the sense that only total functions can be programmed). Reference [Tur95] provides a lively discussion on the merits of such a discipline of typed *total* functional programming. Nevertheless, categorical data types can be interpreted in more structured categories and the program construction principles they entail successfully applied in, for example, languages incorporating partial functions. Such is, typically, the case of HASKELL [HPW92] whose semantics is given upon a category of pointed complete partial orders, imposing a definition ordering on each type. This is an example of an *algebraically compact* category [Fre91] — categories where the initial algebra and final coalgebra of a functor, when existent, are canonnically isomorphic. This ‘collapsing’ of fixpoints makes it possible to encode into a single morphism (said the *hylomorphism*) the common algorithmic principle of *unfold and then fold*, *i.e.*, unfold the argument to populate an intermediate (virtual) data structure and, then, fold over such a structure to build the intended result [MFP91]. On the other hand, however, it obscures, or even prevents, the direct ‘manipulation’ of ‘infinite’, ‘circular’ objects.

15. COALGEBRAIC MODELLING. Mathematically, coalgebras are the formal duals of algebras, and it is exactly in this sense that they are responsible for ‘half the picture’ in categorical data types. As mentioned above, only recently, however, coalgebra theory itself has received enough attention and eventually emerged as a common framework to describe ‘state based’, dynamical, systems. Since then, coalgebraic modelling and reasoning principles have been applied in several areas. Examples range from automata [Rut98] to objects [Rei95, Jac96b], from process semantics [Len98, Sch98, Wol99] to hybrid transition systems [Jac96a].

B. Jacobs and his group, following earlier work by H. Reichel [Rei95, HR95] have coined the term *coalgebraic specification* [Jac97, Jac02] to denote a style of axiomatic specification involving equations up to bisimilarity acting as constraints on system’s observable behaviour. Models of coalgebraic specifications are subcoalgebras of the final coalgebra, just as models for algebraic specifications are quotients of initial algebras. CCSL is a recent proposal of a specification language for classes, including object-oriented structuring mechanisms (*e.g.*, inheritance and aggregation). The language is integrated with a proof assistant. This allows the identification of the functor underlying the specification signature, the automatic generation of the corresponding invariant and bisimulation definitions, as well as assisted proof, *e.g.*, for the verification of specification assertions. Both [RJT01] and [HHJT98] are recent references on this on-going project.

There is a close relationship between coalgebraic specification, and the ‘hidden algebra’ approach referred earlier on §11, as pointed out in, *e.g.*, [Mal96] and [Cir98]. Even though nothing is essentially coalgebraic in ‘hidden algebra’, several constructions (*e.g.*, behavioural equivalence, final models, cofree extensions) can be elegantly regarded as constructions on coalgebras. In particular, the coalgebraic notion of a state space corresponds to (the product of) hidden sorts in hidden-sort algebra. In general, every hidden signature gives rise to an endofunctor whose coalgebras are the hidden models of the signature.

In a broad perspective, coalgebraic modelling explores the close relationship between coalgebras and *modal* and *temporal* logics, whose role in the specification of system’s dynamics is well-established. The basic idea, developed namely in [Mos99] and [Jac99b], consists of associating a modal language to the functor induced by the specification signature, having the logic assertions interpreted over the (Kripke models defined by the) transition systems induced by T-coalgebras.

16. COMPONENTS REVISITED. Further to the context and background just reviewed, one may ask how the notion of a component studied in this thesis fits in the emerging *component-based programming* paradigm [Szy98, MM99]. Rather than a specification framework, such a terminology covers a program development style where reusable off-the-shelf components are glued together to build new applications. Usually components are regarded as *'static abstractions with plugs'* [SN99], implemented and used according to language independent platforms (such as, *e.g.*, COM [Mic92]) and interconnected through *scripting* languages (such as VISUALBASIC, PERL, or even HASKELL, as recently advocated in [LMH98]).

This emerging programming paradigm retains from object-orientation the basic principle of encapsulation of data and code, but shifts the emphasis from (class) inheritance to (object) composition to avoid interference between the former and encapsulation. Thus, a way is paved to a development methodology based on *third-party assembly* of components. The paradigm is often illustrated by the visual metaphor of a *palette* of computational units, treated as black boxes, and a *canvas* into which they can be dropped. Connections are established by drawing *wires*, corresponding to some sort of interfacing code.

As has always been the case with object-orientation (and even with programming in the broadest sense), component-orientation has grown up to a popular technology before consensual definitions and principles, let alone formal foundations, have been put forward. We believe it is not yet a 'paradigm' on its own but rather a collection of technologies and a research concern common to different communities. In 1999, a seminar organised by P. Wadler and K. Weihe was suggestively entitled *Component-Based Programming under Different Paradigms*. The quotation which follows is extracted from the final report [WW99]:

> *The meaning of this word* [component] *is intuitive: programs are broken down into primitive building blocks, which may be flexibly plugged together according to well-defined protocols. In fact, each of the above mentioned programming paradigms* [object-oriented, template-oriented, functional, ...] *may be viewed as an attempt to realise such a component-based programming style. However, the definition of components and the techniques for combining them varies significantly.*

Or, as P. Wadler emphasises in his own contribution to the report, *'just as Eskimos need fifty words for ice, perhaps we need many words for components'*. In such a broad sense, we believe this thesis adds some contribution to a (formal) perspective on the topic.

17. SOFTWARE ARCHITECTURES. Closely related to the emergence of *component-based programming*, the discipline of *software architecture* has arisen as the systematic study of the overall organisation of software systems. In a sense, software architecture scales up 'componentware' in a way similar to what has happened in the hardware field, as the size and complexity of computer systems increased.

During the last decade, a number of organisational patterns have been recognised in the software design practice — just recall expressions like *'client-server'*, *'layered model'*, *'pipeline architecture'*, *'blackboard system'*, *'object community'* and so on. Reference [GS93] provides a tutorial introduction to the area, from a qualitative point of view, and [Gog96] is pure 'food for thought'.

We are, however, far away from having a suitable taxonomy of architectural paradigms; let alone a mathematical theory. What seems to be reasonably consensual is the need to describe an architecture in terms of collections of *components*, in one hand, and *connectors* (which may turn out to be just special kinds of components), on the other. Some formal description notations have been proposed. The WRIGHT methodology [AG97], for example, resorts to a process notation (a subset of CSP [Hoa85]) to describe both components' behaviour at specific ports and the *glue* specification at connector level. This combines the behavioural patterns expected for any components willing to be connected using that specific connector. The emphasis is put on maximising *reuse* to which, as one could expect, corresponds the technical problem of formally defining and ensuring static and dynamical *compatibility*.

There is a variety of approaches dealing with similar problems in a sound, formal way. The literature includes references such as [MQ94], [Nie93] (the latter restricted to an object-oriented framework) and [MDEK95], [SN99], both based on the π*-calculus* [Mil99], as well as recent work by J. Fiadeiro and collaborators at Lisbon (see, *e.g.*, [FL97, WF98, WLF01]).

3. Outline of Contents and Contributions

18. OVERVIEW. As stated in §3, this thesis intends to propose a semantical model for *software components*, parametric on a notion of behaviour, and to develop a *calculus* to reason about component based designs.

Regarded as state-based, dynamical systems, components are modelled as coalgebras for a class of endofunctors in Set, with specified initial conditions. Two basic classes of functors, differing on the degree in which output may depend on input, are considered. Each of them entails a notion of bisimilarity and gives rise to a family of calculi parametrized by a strong monad which captures a particular behaviour model. Partiality and nondeterminism are typical examples, but, of course, not the only ones.

Components are, thus, *concrete* coalgebras, over a state space supporting a set of observers and actions which are explicitly specified. For each inhabitant of the state space, the corresponding behaviour, or 'process', arises by coinductive extension (*i.e.*, as its anamorphic image). In fact, *processes*, but not *components*, can be regarded as inhabitants of a final coalgebra. Coinductive types provide an abstraction for *behaviours*, just as inductive types characterise *data*. Both of them, as canonical representatives of, respectively, coalgebraic and algebraic structures, can be formally manipulated in computer programs, provided the programming language has enough expressive power. Animating components' behaviour in a language supporting coinductive types, is a recurrent theme along the thesis.

Regarding behaviours as 'duals' to data objects such as, say, finite lists or trees, raises a related question also addressed in the thesis: process calculi, being about a particular sort of behaviours, could they be developed and their laws proved along the lines one gets used to in (data-oriented) program calculi?

19. CONTRIBUTIONS. The thesis main contributions are organised in two areas:

- The development of CCS-like process calculi in a *generic* framework whereby reasoning is carried out in a *calculational* style.
- The study of *components* as concrete coalgebras and the development of the corresponding calculi, again in a *generic* framework.

These two areas share the same (coalgebraic) modelling discipline and a common concern, which we would like to see as the driving *motto* of the whole work: *genericity*. This means that the proposed models and calculi are *parametric* with respect to the underlying *interaction discipline* (in the case of processes) and the *behaviour model* (in that of components).

The way this is achieved in the thesis is sketched in the following paragraphs which detail its contents. A somewhat more detailed discussion of the thesis contributions, in comparison with the literature, is, however, delayed to the concluding chapter, after the complete presentation of the work. Should the reader prefer to see the thesis contributions put in perspective right now, a quick visit to the first section of chapter 7 would be the natural complement to this introduction.

20. ORGANISATION. Chapters 2 and 3 as well as appendixes A, B and E provide the background material for the thesis.

Processes as *codata* are studied in chapter 4. The chapter, however, begins with a brief 'exercise' on coalgebraic modelling and its last section introduces the component models and discusses their relationship to processes. This acts as a 'bridge' to the following chapters.

The development of calculi for components characterised by either a 'functional' or 'object-like' shape, is the main subject of chapters 5 and 6, respectively. In both cases genericity is achieved by the monadic parametrization mentioned above. In order to correctly handle such a parametrization, a pointfree calculational proof style is adopted. This leads to easy to follow (but sometimes long) proofs, most of them collected in appendix D for increased readability. Appendix C, on the other hand, states and proves a number of laws relating common 'housekeeping' morphisms (such as associativity) with monad's unit, multiplication and strength. Such 'context laws' are required by several proofs of components' properties and constitute a sort of 'side-product' of this thesis.

Finally, chapter 7 concludes, compares and introduces some research topics for future work. A particular mention is made to component *refinement*.

A 'roadmap' to the themes addressed in the thesis is suggested in the following paragraphs.

21. PROCESSES AS CODATA. Chapter 4 is an attempt to develop process calculi on top of a representation of processes as inhabitants of coinductive types, *i.e.*, final coalgebras for suitable Set endofunctors. Most of this chapter is concerned with functor

$$\mathcal{P}(Act \times \mathsf{Id})$$

which gives rise to a family of CCS-like calculi. The set Act of actions is equipped with the structure of a positive monoid with a zero element, which specifies an *interaction* discipline. Different calculi arise by varying either this monoid (leading to different interaction disciplines within the same behaviour model) or the process structure itself, replacing $\mathcal{P}$ by another monad. This leads to different behaviour paradigms under the same interaction discipline.

Final semantics for processes is an active research area, namely after Aczel's landmark paper [Acz93]. Our emphasis is, however, placed on the design side: we intend to define process combinators just as (dually to, to be exact) we build operations on binary trees, and to reason about them in the same calculational style. Placing data and behaviour at a similar level conveys the idea that process models can be chosen and specified according to a given application area, in the same way that a suitable data structure is defined to meet a particular engineering problem. Moreover it leads to a *generic* approach to process calculi design and entails a purely calculational (basically *equational* and *pointfree*) proof style. This circumvents the explicit construction of bisimulations used in most of the literature on process calculi. In particular, a 'conditional fusion' theorem is proved and its use is illustrated in the derivation of conditional laws. Part of this work appeared in [Bar01a].

22. COMPONENTS AS ARROWS. Following a fundamental intuition of functional programming, software components are considered in chapter 5 as *arrows* between two observation universes, thought of as input and output interfaces. Functions from, say, I to O are written as $f : I \longrightarrow O$ and defined as inhabitants of the exponential type, *i.e.*, $f \in O^I$. Similarly, components will be written as $p : I \longrightarrow O$. But how should we 'fill in the dots' in expression $p \in \cdots$?

Such suitably typed arrows should compose and the resulting construction should have, at least, the structure of a category. The shared observational universe involved in a composition situation acts, as it actually does in functional composition, as an *interaction* space.

Filling in the dots, components $p : I \longrightarrow O$ will be introduced in this chapter as *seeded concrete coalgebras* for the following family of Set endofunctors

$$\mathsf{T}^{\mathsf{B}} \;=\; \mathsf{B}\,(\mathsf{Id} \times O)^I$$

where B is a strong monad intended to capture a particular behavioural model.

In order to deal with static, as well as dynamic, aspects of components, we will have to consider arbitrary coalgebras defined over concrete state spaces, instead of restricting ourselves to work within final coalgebras. This decision stems from our starting focus on model-oriented specification methods: we want to deal explicitly with the structure of the state spaces, to compare them and, eventually, to discuss the effects of their transformations in the overall component behaviour. There is however a price to be paid: the presence of concrete state spaces makes the usual axioms for a category only verifiable up to isomorphism, ending up with the looser structure of a *bicategory* [Ben67].

Therefore, a bicategory Cp_{B}, with interfaces as objects, components and component morphisms as arrows and 2-cells, respectively, is defined where composition amounts to a pipe-like interconnection of independent components. A more general interaction scheme is captured by total and partial feedback operators. Several other aggregation combinators, a wrapping mechanism and the lifting of functions to Cp_{B}

are studied. All the definitions and the resulting calculus are parametric with respect to the underlying behavioural model, relying only on standard properties of the underlying monad, namely strength and, for a few cases, commutativity. Our approach is in debt to the work of R. Walters and his collaborators on 'bicategories of processes' (see §7), which we generalise in a sense to be discussed later in §7.5.

A category Bh_{B} of *behaviours* is derived from Cp_{B} by reducing objects of its hom categories to their anamorphic images in the final T^{B}-coalgebra. Cp_{B} hom categories become simply hom sets, but the component algebra lifts naturally to this setting, now as a 'process calculus': its laws become valid up to equality. This derived category will play an important role in the overall framework as a space for *prototyping* (§25).

23. SEPARABLE COMPONENTS. Component calculi developed in the thesis are concerned with component interconnection and interaction, but they are 'blind' with respect to their internal structure. Being aware of some details of such a structure — for example, some properties of the state space or the specification format for the coalgebra dynamics — enables a finer 'tuning' of the calculi. In chapter 5, a particular class of components, called *separable*, are studied. A separable component is specified as a collection of actions over a shared state space, each of them with a specific interface. Packing them into a T^{B}-coalgebra, results in an additive interface such that each input stimulus produces a result whose type is known and unique. Most components defined in model-oriented frameworks fall in such a class. It is shown how this restriction entails a richer ontology of interaction mechanisms captured by new feedback combinators.

24. COMPONENTS AS OBJECTS. If chapter 5 regards components as '(monadic) functions with memory', chapter 6 adopts an 'object-oriented' perspective. In particular, observers (or 'attributes') are unrelated to actions (or 'methods') and can, therefore, be accessed in parallel. In this way, components $I \xrightarrow{p} O$ become coalgebras for

$$\mathsf{T}^{\mathsf{B}} \;=\; O \times \mathsf{B}^{I}$$

again parametric on a behaviour strong monad B.

The component calculus developed in chapter 5 is revisited in this new setting. Although the definitions of the combinators have to be adjusted, most of the previous results remain valid. Moreover, interaction, modelled by feedback combinators which connect a 'multiplicative' output to an 'additive' input, becomes easier to deal with as feedback parameters can be hidden.

There is, however, a fundamental difficulty concerning the definition of an appropriate bicategory as (sequential, or pipelining) composition fails to be associative. This justifies the introduction of a broader definition of a component morphism (called a *next comorphism*) entailing an equally broader notion of bisimilarity. The seed value considered in the 'functional' model of chapter 5 has also to be replaced by a seed *predicate*, *i.e.*, a collection of seed values.

Having succeeded in defining a new bicategory of components, the calculus is, however, introduced in a derived category — St_{B} —- in which components are objects (another justification for the chapter's name!). The motivation for this is to avoid the technical restriction to next comorphisms, given that several, although not all, the relevant laws can be given as straight bisimilarity equations.

Last but not least, this paves the way to a further generalisation of component morphisms. The basic idea is to relate components with different types at the level of morphisms, these being defined up to a natural transformation $\tau_{f,g}$ encoding interface conversion. Transformation $\tau_{f,g}$ is determined by functions f on the output and, contravariantly, g on the input. Moreover, g can be any arrow in the Kleisli category for the behaviour monad B, *i.e.*, g will be typed as $g : I' \longrightarrow \mathsf{B}\ I$ instead of being just a function from I' to I. This means that extra structure in the input conversion can be absorbed by the underlying behaviour model. Some properties of such morphisms are proved. In particular, the resulting category is shown to be cofibred over the interface space.

Another element considered in this chapter is the incorporation of *internal activity*, independent of the component's environment. This is done in a simple way, conveying the intuition that, when interacting with a component, the current value of its state space may be different from the one left by a former interaction. Part of the results in this chapter appeared in [Bar00].

25. PROTOTYPING. Along chapters 4, 5 and 6, we resort to the experimental programming language CHARITY [CF92] to prototype some of the proposed constructions, namely the process calculi and the component algebras studied. As a programming language, CHARITY is based on the term logic of distributive categories and provides a definitional mechanism for categorical data types, *i.e.*, for both initial algebras and final coalgebras. The latter are regarded in this thesis as 'behavioural structures': behaviours are, in fact, elements of Bh_{B} and we are therefore given a tool to animate components' evolution and interconnection.

On the other hand, we believe that prototyping plays an important role in any specification method, as it supports a stepwise development style. With such a tool each design stage can be immediately animated and quick feedback about its behaviour gathered. Functional programming languages have been a favourite vehicle for rapid prototyping of formal specifications, at least since P. Henderson's me too [Hen84] proposal for animating VDM. Should our intention be not limited to prototyping system's static functionality, a language with coinductive data types offers a new level of experiments.

26. CONTEXT LAWS. As mentioned above, appendix C states and proves several generic laws relating 'housekeeping' morphisms with monad's unit, multiplication, strength and strength distribution. They are essential to most proofs of the components' laws. And we would like to regard them them as an 'add-in' to the 'pointfree calculator toolbox'.

In most cases we have proceeded on a 'demand driven' basis. Albeit being non exhaustive, we believe they may be useful in other situations of context manipulation in expressions involving strength and the monad definitional morphisms.

27. BACKGROUND. Chapters 2 and 3, as well as appendixes A, B and E, provide some background material for the whole thesis. Chapter 2 is an introduction to basic category theory, complemented in appendixes A and B with a review of some specific concepts and results on monads and bicategories, respectively. Chapter 3, on its turn, introduces algebras and coalgebras for a functor. It also makes an incursion into the categorical approach to data types, as a framework in which both data structures and behavioural patterns can be abstracted, as, respectively, initial algebras and final coalgebras.

In retrospect, we have to admit that a thick time slice of our PhD work was devoted to study and, hopefully, to master, a number of core concepts and techniques in these areas. Their account in these chapters of the thesis is, of course, not original, but reflects our own perception of them and, even our learning process. This can be seen in the overall organisation and the introductory motivation to chapter 3.

The final section of chapter 3 attempts to provide an introduction to strong data types and the corresponding universals, along the same lines the 'non strong' picture is previously reviewed. The outcome is slightly different from what can be found in the 'standard' references, namely in the CHARITY accompanying papers [CS92, CS95]. The latter resort to a fibrational setting which is probably less familiar to a computer science audience.

Finally, a brief introduction to the CHARITY language is provided in appendix E, which includes an exercise in defining sets and partial functions, and some associated operators, as higher-order coinductive types.

28. A REMARK ON NOTATION. Elementary category theory will be used throughout this thesis to formulate and organise results. In particular, Set, the category of sets and set-theoretic functions, is assumed as the underlying universe.

The notation is hopefully clear and standard. However, a few remarks are in order. First note that we explicitly denote composites of arrows and vertical composition of natural transformations (more generally, of 2-cells in a bicategory) by $f \cdot g$. Horizontal composition of natural transformations (more generally, of 2-cells) is denoted by $g \mathbin{;} f$ and used in diagrammatic order. Expression parentheses are omitted wherever possible (their use is to disambiguate expressions). In particular, the application of morphisms to arguments is denoted by juxtaposition and angle brackets are used to represent tuples. In some occasions, lambda expressions are used to denote elements of exponentials.

A final convention: binary connectives always associate to the left and morphism application binds more tightly than composition. Therefore, an expression $x \otimes y \otimes z$ stands for $(x \otimes y) \otimes z$. Likewise, $\mathsf{T}\ (f \cdot g)$ and $\mathsf{T}\ f \cdot g$ are different morphisms.

This thesis is organised in paragraphs, each of which addresses a single topic. Every definition, lemma or simple remark, appears in an independent paragraph. Paragraph's are numbered sequentially within each chapter. A reference to paragraph n of chapter i is indicated by $\S n$, if it is placed *inside* that chapter, or by $\S i.n$ otherwise.

CHAPTER 2

Categorical Preliminaries

Summary

It is remarkable that so many computational phenomena and concepts in programming semantics find a categorical 'explanation' which is both concise and generic. Software components, studied in the subsequent chapters, turn out to be one of them, as emphasised in the title of this thesis. This chapter provides a brief introduction to some concepts of elementary category theory, up to the notion of an adjunction. It also introduces the basic notation used throughout the thesis.

1. Basic Notions

1. INTRODUCTION. In the introductory material of [McL92], Colin McLarty comments on the development of category theory as follows:

> *The spread of applications led to a general theory, and what had been a tool for handling structures became more and more a means of defining them. (...) In the 1960s, Lawvere began to give purely categorical descriptions of new and old structures, and developed several styles of categorical foundations for mathematics. This lead to new applications, notably in logic and computer science.*

The notion of a *coalgebra*, which pervades this thesis, is one of such old-new structures, easily recognised in a variety of computational phenomena, but whose generality and expressive power became clear only under the light of category theory. This chapter offers an introduction to basic category theory by presenting some core concepts, notation and laws. Algebras and coalgebras for an endofunctor and their 'canonical' representatives, known as categorical datatypes, will be introduced later, in chapter 3. Two topics not covered in the main text, namely *monads* and *bicategories*, are deferred to appendixes A and B, respectively, for easier reference and to avoid a too lengthy background chapter.

Our exposition of background material expresses a personal way of understanding the basic concepts and results reviewed. The reader is referred to standard textbooks, namely [Mac71] and [Bor94a], for proofs and a more detailed exposition. [BW90a] and [Wal91] are intended for a computer science audience, but several other introductory texts exist. One we have found particularly sharp and clear is [McL92]. On the other hand, to build up one's own intuitions, [LS97] remains a 'conceptual' pearl.

2. CATEGORIES. Roughly speaking, categories deal with *arrows* and their composition, in the same sense that sets deal with *elements*, their aggregation and membership. An *arrow* is an abstraction of the familiar notion of a function in set theory or of a homomorphism in algebra. Depicted as $f : X \longrightarrow Y$, it may be thought of as a transformation, or, simply, a connection, between two *objects* X and Y, called its *source* (or domain) and *target* (or codomain), respectively. The sources and targets of all the arrows in a category, form the class of its *objects*. If the same object is both the target of an arrow f and the source of another arrow g, f and g are said to be composable. Arrow composition is thus a partial operation and what the axioms for a category say is that arrows and arrow composition form a sort of generalized monoid. Formally,

3. DEFINITION. A *category* C consists of

- a class $\mathsf{obj}(\mathsf{C})$ of *objects* $X, Y, Z, ...$
- for each pair of objects X and Y, a set of *arrows* with source X and target Y, called a *hom set* and denoted by $\mathsf{C}[X, Y]$.
- for each triple of objects X, Y and Z, an operation, called C-*composition*,

$$\cdot : \mathsf{C}[Y, Z] \times \mathsf{C}[X, Y] \longrightarrow \mathsf{C}[X, Z]$$

such that

- C-composition is *associative* and there exists, for each object X, a special arrow id_X (or simply X if no confusion with the object X itself arises), called the *identity on* X, which is both a pre- and post-unit of composition. This is neatly expressed by the commutativity of the following diagrams:

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[dr, "g \cdot f"'] & Y \arrow[d, "g"] \arrow[dr, "h \cdot g"] & \\
& Z \arrow[r, "h"] & W
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, "f"] \arrow[dr, "f"'] & Y \arrow[d, "\mathsf{id}_Y"] \arrow[dr, "g"] & \\
& Y \arrow[r, "g"] & Z
\end{tikzcd}$$

i.e., by

$$\begin{aligned} h \cdot g \cdot f &= h \cdot (g \cdot f) && (2.1)\\ \mathsf{id}_Y \cdot f &= f && (2.2)\\ g \cdot \mathsf{id}_Y &= g && (2.3) \end{aligned}$$

- Arrows uniquely determine their source and target objects, *i.e.*, for all X and Y, the sets $\mathsf{C}[X, Y]$ are pairwise disjoint.

4. EXAMPLES. A prime example of a category is Set, the category of sets and set-theoretic functions. Another example is Rel in which functions are replaced by binary relations. Composition is then relational product: given arrows $r \subseteq Y \times Z$ and $s \subseteq X \times Y$, their composite is the relation $r \circ s = \{\langle x, z\rangle \subseteq X \times Z | \ \exists_{y \in Y} \, . \, \langle x, y\rangle \in s \wedge \langle y, z\rangle \in r\})$. Other typical examples of a category arise by considering algebraic structures and their homomorphisms or, even, by looking at a particular such structure as a category itself. For example, a poset forms a category by taking as arrows all the pairs of elements x, y such that $x \leq y$. Composition and identities are given, respectively, by the transitivity and reflexivity of the order relation.

There are also several standard ways of obtaining new categories from old. The simplest one consists of formally reversing all the arrows of a category C. The result, denoted by C^{op}, is referred to as the *dual category* of C. Another useful construction is the *product category*: given two categories C and D, their product category has as objects (resp. arrows) pairs $\langle C, D\rangle$ (resp. $\langle f, g\rangle$) of a C and a D-object (resp. arrow). Identities and composition are defined componentwise.

5. UNIVERSALITY. If there is a ‘main topic’ in category theory, this is certainly the study of *universal* properties. Roughly speaking, an entity ϵ is universal among a family of ‘similar’ entities if it is the case that every other entity in the family can be *reduced* or *traced back* to ϵ. For example, an object T is said to be *final* in a category C if, from every other object X in C, there exists a unique arrow $!_X$ to T. Therefore, there is a canonical, in the sense of *unique*, way to relate every object in C to T — *finality* is thus an universal property.

A nice thing about universal properties is the fact they always ‘come in pairs’: the *dual* of an universal is still an universal. Dualizing finality, we arrive at *initiality*: an object is *initial* in C if there is one and only one arrow in C from it to any other object in the category.

Universal properties, like finality or initiality, can be recognised, usually under a different terminology, in many branches of Mathematics. Moreover, they happen to play a major role in the structure of ‘mathematical spaces’. Therefore, category theory provides a setting for studying abstractly such ‘spaces’ and their relationships.

6. Final and Initial Objects. In Set the empty set has exactly the properties of an initial object. On the other hand, if we seek for final objects, we will end up recognising that any *singleton* set will do. The usual notation for the empty set is $\emptyset$, a symbol close to 0. To stress the duality, any (actually, the) singleton will be denoted by $\mathbf{1}$. The same symbols will be used for initial and final objects in any category. The corresponding universal properties state the existence, for each object X in the category, of two unique arrows $!_X : X \longrightarrow \mathbf{1}$ and $?_X : \emptyset \longrightarrow X$ such that, for any $f : X \longrightarrow Y$:

$$!_Y \cdot f \; = \; !_X \tag{2.4}$$

$$f \cdot ?_X \; = \; ?_Y \tag{2.5}$$

7. Points. One way of thinking of an arrow $x : Z \longrightarrow X$ is as an ‘element’ of X, which is not given once and for all, but depends on Z. Following [McL92], we shall call x a *generalized element* of X and Z its *stage of definition*. This suggests the alternative notation $x \in_Z X$ for arrow $x : Z \longrightarrow X$. The composite $f \cdot x$, for $f : X \longrightarrow Y$, can thus be written as $f\ x$.

A special kind of elements of an object X consists of arrows into X whose source is the final object $\mathbf{1}$ (§6) in the category (if it exists). They are called *points* (or *global elements*) of X. In some categories every arrow $f : X \longrightarrow Y$ is fully determined by its effect on the points of X. Should this be the case, the category is said to be *well pointed*. Again Set is a good example: being well pointed is just a categorical version of the well known fact that ‘a set is determined by its elements’. In Set, which will be the working universe in this thesis, we shall make explicit the correspondence between elements $x \in X$ and points $\underline{x} : \mathbf{1} \longrightarrow X$, by denoting function application by juxtaposition, *i.e.*, $fx = f \cdot \underline{x}$.

8. Isomorphism. What is ‘inside’ $\mathbf{1}$? More generally, in what sense an universal entity is, in fact, *the* universal? In category theory, elements and extensionality have been abstracted away and therefore the internal structure of C objects is not available when reasoning at the level of C. Objects that cannot be distinguished in the language of category theory are called *isomorphic*. Think, for a moment in sets like $\{0\}$, $\{20\}$, $\{\texttt{abc}\}$ or $\{*\}$. From a categorical point of view they have to be characterised in terms of their behaviour under ingoing or outgoing arrows. And it turns out that all that can be said of them, from this point of view, is that they fulfil the properties of a final object. As it is not possible to distinguish them further, the symbol **1** has been taken to denote their isomorphism class. In other words, we may say that each such set is the final element in Set *up to isomorphism*. In fact, all objects defined by universal constructions are unique up to isomorphism.

9. DEFINITION. The very notion of isomorphism is stated in the language of arrows. An arrow $f : X \longrightarrow Y$ is an *isomorphism* if there is another arrow $g : Y \longrightarrow X$ such that

$$f \cdot g = \mathsf{id}_Y \tag{2.6}$$

$$g \cdot f = \mathsf{id}_X \tag{2.7}$$

If such a g exists it is said the *inverse* of f and written f°. A right (resp. left) inverse to f is also called a *section* (resp. *retraction*). If there is an isomorphism between two objects X and Y, they are said to be *isomorphic* and we write $X \cong Y$.

10. CANCELLATION. A weaker requirement an arrow may satisfy is *cancellation*. Like existence of inverse, it has a right and a left version. We say $f : X \longrightarrow Y$ is *cancellable on the right*, an *epimorphism*, or, simply, an *epi*, if, for every $z_1, z_2 : Y \longrightarrow Z$,

$$z_1 \cdot f = z_2 \cdot f \;\; \Rightarrow \;\; z_1 = z_2 \tag{2.8}$$

Dually, it is said to be *cancellable on the left*, a *monomorphism*, or, simply, a *mono*, if, for every $x_1, x_2 : Z \longrightarrow X$,

$$f \cdot x_1 = f \cdot x_2 \;\; \Rightarrow \;\; x_1 = x_2 \tag{2.9}$$

Cancellation propagates through composition, in the sense that, in any category, any composition of monos (resp. epis) is still a mono (resp. an epi).

11. MONOMORPHISMS. If we think of x_1 and x_2 as generalized elements of X (§7), the definition of a *mono* above looks familiar: for all, $x_1, x_2 \in_Z X$,

$$f x_1 = f x_2 \;\; \Rightarrow \;\; x_1 = x_2 \tag{2.10}$$

resembles the definition of an injective function. In fact, a *mono* is an injection over generalized elements and happens to be an injection in Set and in most categories of sets with structure.

12. EPIMORPHISMS. On the other hand, an *epi* does not convey entirely the notion of a surjection. In fact, what the definition of an epi $f : X \longrightarrow Y$ says is that f covers a 'sufficiently large' part of Y, in the sense that, to be distinguished, arrows from Y must disagree somewhere in the part covered by f. The stronger property of being right invertible (2.6) is required to capture surjectivity. In the language of generalized elements, the latter reads,

$$\text{for each } Z \text{ and } y \in_Z Y\text{, there exists an } x \in_Z X \text{ such that } f\ x = y$$

One can easily show that a right invertible arrow is also an epi, said a *split epi*. In Set, but not in many other cases, all epis are split.

13. FUNCTORS. Once a structure has been introduced, the natural next step one gets used to from Universal Algebra, is to look for an appropriate definition of a morphism that preserves such a structure. A *functor* is exactly such a morphism for categories (*i.e.*, a homomorphism of categories). Therefore, it preserves the typing of arrows and identities and distributes over composition. Formally,

14. DEFINITION. A *functor* $\mathsf{T} : \mathsf{C} \longrightarrow \mathsf{D}$ from a category C to a category D is a mapping assigning to each $X \in \mathrm{obj}(\mathsf{C})$ an object $\mathsf{T}\ X$ in $\mathrm{obj}(\mathsf{D})$ and, similarly, to each $f \in \mathsf{C}[X, Y]$ an arrow $\mathsf{T}\ f \in \mathsf{D}[\mathsf{T}\ X, \mathsf{T}\ Y]$ such that

$$\mathsf{T}\ \mathsf{id}_X = \mathsf{id}_{\mathsf{T}\ X} \tag{2.11}$$

$$\mathsf{T}\ (f \cdot g) = \mathsf{T}\ f \cdot \mathsf{T}\ g \tag{2.12}$$

15. CAT. As expected, functors with compatible typing can be composed. Functor composition will be represented in the sequel by $\circ$ or, more often, by juxtaposition. Moreover, for each category C there is a an identity functor denoted by C or Id_{C} (or simply Id) which is the identity on both objects and arrows. Therefore, categories and functors form themselves a category Cat.

16. SPECIAL FUNCTORS. A functor from a category C to itself is said to be an *endofunctor*. On the other hand, a functor whose source is a product category is called a *bifunctor* and is often represented by an infix operator. Given a bifunctor $\mathsf{T} : \mathsf{C} \times \mathsf{D} \longrightarrow \mathsf{E}$ and an object C of C, a *C-section* of T is the functor $\mathsf{T}_C : \mathsf{D} \longrightarrow \mathsf{E}$ obtained from T by fixing its first argument.

Associated with each object $X \in \mathrm{obj}(\mathsf{C})$, there is also the *constant* functor on X which maps every object to X and every morphism to the identity on X. This functor will be simply written X (or $\underline{X}$, if the context is ambiguous).

17. NATURAL TRANSFORMATIONS. Functors can be seen not only as arrows in Cat, but also as objects of other categories, provided a suitable notion of morphism is defined. This is exactly what a *natural transformation* is. Formally,

18. DEFINITION. Given two functors $\mathsf{T}, \mathsf{S} : \mathsf{C} \longrightarrow \mathsf{D}$ a *natural transformation* $\sigma : \mathsf{T} \Longrightarrow \mathsf{S}$ is a family of D-arrows, indexed by the objects of C, such that, for any C-arrow $f : X \longrightarrow Y$ the following diagram commutes:

$$\begin{tikzcd}
X \arrow[d, "f"'] & & \mathsf{T}\,X \arrow[r, "\sigma_X"] \arrow[d, "\mathsf{T}\,f"'] & \mathsf{S}\,X \arrow[d, "\mathsf{S}\,f"] \\
Y & & \mathsf{T}\,Y \arrow[r, "\sigma_Y"'] & \mathsf{S}\,Y
\end{tikzcd}$$

i.e.,

$$\sigma_Y \cdot \mathsf{T}\,f \ = \mathsf{S}\,f \cdot \sigma_X \tag{2.13}$$

Each σ_X is referred to as the *component* of σ at the object X.

19. VERTICAL COMPOSITION. Suppose T, S and R are functors from C to D and that there are natural transformations $\sigma : \mathsf{T} \Longrightarrow \mathsf{S}$ and $\sigma' : \mathsf{S} \Longrightarrow \mathsf{R}$. Then, σ and σ' can be composed originating $\sigma' \cdot \sigma : \mathsf{T} \Longrightarrow \mathsf{R}$, by defining

$$(\sigma' \cdot \sigma)_X \ = \sigma'_X \cdot \sigma_X \tag{2.14}$$

This is known as the *vertical* composition of natural transformations. Thus, we may form the *functor category*, D^{C}, of functors from C to D and natural transformations. Notice that, for each functor T in D^{C}, the family of identity arrows $\mathsf{id}_{\mathsf{T}\,X}$ in D gives rise to a trivial natural transformation denoted by 1_{T}, which acts as an identity in D^{C}.

20. HORIZONTAL COMPOSITION. There is also a notion of composition for natural transformations between pairs of composable functors. It will be denoted by by **;** and used in diagrammatic order. Suppose T and S are functors from C to D and T' and S' are functors from D to E. If there exist natural transformations $\sigma : \mathsf{T} \Longrightarrow \mathsf{S}$ and $\sigma' : \mathsf{T}' \Longrightarrow \mathsf{S}'$, their *horizontal* composite is $\sigma \,;\, \sigma' : \mathsf{T}'\,\mathsf{T} \Longrightarrow \mathsf{S}'\,\mathsf{S}$ whose component at X is given by

$$(\sigma \,;\, \sigma')_X \ = \mathsf{S}'\,\sigma_X \cdot \sigma'_{\mathsf{T}\,X} \ = \ \sigma'_{\mathsf{S}\,X} \cdot \mathsf{T}'\,\sigma_X \tag{2.15}$$

as, by definition of σ and σ', the following diagram commutes:

$$\begin{tikzcd}
\mathsf{T}'\,\mathsf{T}\,X \arrow[r, "\sigma'_{\mathsf{T}\,X}"] \arrow[d, "\mathsf{T}'\,\sigma_X"'] \arrow[dr, "(\sigma;\sigma')_X"] & \mathsf{S}'\,\mathsf{T}\,X \arrow[d, "S'\,\sigma_X"] \\
\mathsf{T}'\,\mathsf{S}\,X \arrow[r, "\sigma'_{\mathsf{S}\,X}"'] & \mathsf{S}'\,\mathsf{S}\,X
\end{tikzcd}$$

Particular cases of this situation occur when σ or σ' are the identity 1_{R} on a functor R. We may, then, pre- or post-compose σ with 1_{R}, yielding

$$\mathsf{R}\sigma \overset{\text{abv}}{=} \sigma\,;1_{\mathsf{R}} : \mathsf{R}\,\mathsf{T} \longrightarrow \mathsf{R}\,\mathsf{S} \qquad \text{with} \quad (\mathsf{R}\sigma)_X = \mathsf{R}\,\sigma_X \tag{2.16}$$

$$\sigma\mathsf{R} \overset{\text{abv}}{=} 1_{\mathsf{R}}\,;\sigma : \mathsf{T}\,\mathsf{R} \longrightarrow \mathsf{S}\,\mathsf{R} \qquad \text{with} \quad (\sigma\mathsf{R})_X = \sigma_{\mathsf{R}\,X} \tag{2.17}$$

where $\mathsf{T}, \mathsf{S} : \mathsf{C} \longrightarrow \mathsf{D}$, $\sigma : \mathsf{T} \Longrightarrow \mathsf{S}$ and R is a functor from D to E, in the first case, and from B to C, in the second. Vertical and horizontal composition of natural transformations interact via the *interchange law* (see §B.4). In fact, natural transformations are 2-cells (§B.7) in a particularly common 2-category, as detailed in appendix B.

2. Universality and Calculation

21. From a programming point of view, it is remarkable that the basic properties captured in the categorical framework — such as *universality*, *functoriality* and *naturality* — can be phrased in a 'calculational' style. This means that such properties can be formulated as (usually equational) laws and used to manipulate and reason about objects and arrows of the underlying category. Such a 'calculational' style matches nicely a main concern in computer science — the seek for program calculi able to promote programming to a modern engineering discipline.

In the next paragraphs we will recall the *product* and *coproduct* constructions and some associated laws that turn out to be most useful in calculation. Product and coproduct are the categorical generalisation of Cartesian product and disjoint union in Set. In a sense, they capture the duality between *co-occurrence* and *choice*, which may explain their major role in modelling computational systems. From §24 on, universal constructions, like product and coproduct, or final and initial object, will be revisited in this broader, more comprehensive setting.

22. PRODUCT. The *product* of two objects X and Y in a category C is an object $X \times Y$ defined as the source of two arrows $\pi_1 : X \times Y \longrightarrow X$ and $\pi_2 : X \times Y \longrightarrow Y$, called the *projections*, which satisfy the following universal property: for any other $Z \in \mathsf{obj}(\mathsf{C})$ and arrows $f : Z \longrightarrow X$ and $g : Z \longrightarrow Y$, there is a unique arrow $\langle f, g \rangle : Z \longrightarrow X \times Y$, usually called the *split* of f and g, that makes the following diagram to commute:

$$\begin{tikzcd}
& Z \arrow[ld, "f"'] \arrow[d, "\langle f,g\rangle"] \arrow[rd, "g"] & \\
X & X \times Y \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & Y
\end{tikzcd}$$

This universal property can be written as

$$k = \langle f, g \rangle \;\Leftrightarrow\; \pi_1 \cdot k = f \;\wedge\; \pi_2 \cdot k = g \tag{2.18}$$

where $\Rightarrow$ means *existence* and $\Leftarrow$ means *uniqueness*. Form (2.18) the following laws are easily derived:

$$\pi_1 \cdot \langle f, g \rangle = f \;,\; \pi_2 \cdot \langle f, g \rangle = g \tag{2.19}$$

$$\langle \pi_1, \pi_2 \rangle \;=\; \mathsf{id}_{X \times Y} \tag{2.20}$$

$$\langle g, h \rangle \cdot f \;=\; \langle g \cdot f, h \cdot f \rangle \tag{2.21}$$

which exemplify, for the product construction, what is sometimes called a *cancellation*, *reflection* and *fusion* result, respectively. Structural equality is also derivable from (2.18):

$$\langle f, g \rangle = \langle k, h \rangle \;\equiv\; f = k \wedge g = h \tag{2.22}$$

23. COPRODUCT. The *sum*, or *coproduct*, of X and Y in a category C is the dual construction — actually it simply is the product in C^{op}. That is to say, an object $X + Y$ defined as the target of two arrows $\iota_1 : X \longrightarrow X + Y$ and $\iota_2 : Y \longrightarrow X + Y$, called the *injections*, which satisfy the following universal property: for any other $Z \in \mathsf{obj}(\mathsf{C})$ and arrows $f : X \longrightarrow Z$ and $g : Y \longrightarrow Z$, there is a unique arrow $[f, g] : X + Y \longrightarrow Z$, usually called the *either* (or *case*) of f and g, that makes the following diagram to commute:

$$\begin{tikzcd}
X \arrow[r, "\iota_1"] \arrow[dr, "f"'] & X+Y \arrow[d, "{[f,g]}"] & Y \arrow[l, "\iota_2"'] \arrow[dl, "g"] \\
 & Z &
\end{tikzcd}$$

Again this universal property can be written as

$$k = [f,g] \quad \Leftrightarrow \quad k \cdot \iota_1 = f \ \wedge \ k \cdot \iota_2 = g \tag{2.23}$$

from which one infers correspondent *cancellation*, *reflection* and *fusion* results:

$$[f,g] \cdot \iota_1 = f \ , \ [f,g] \cdot \iota_2 = g \tag{2.24}$$

$$[\iota_1, \iota_2] \ = \ \mathsf{id}_{X+Y} \tag{2.25}$$

$$f \cdot [g,h] \ = \ [f \cdot g, f \cdot h] \tag{2.26}$$

Products and sums interact through the following *exchange* law

$$[\langle f,g \rangle, \langle f',g' \rangle] \ = \ \langle [f,f'], [g,g'] \rangle \tag{2.27}$$

provable by either × (2.18) or + (2.23) universality.

24. Universality Revisited. If categories can be thought of as particular mathematical spaces and functors as structure-preserving translations between them, an *adjunction* between, say, two functors $\mathsf{T} : \mathsf{C} \longrightarrow \mathsf{D}$ and $\mathsf{S} : \mathsf{D} \longrightarrow \mathsf{C}$, can be regarded as a source of *universals* in C and D. In fact, products and coproducts, final and initial objects and, in general, any universal construction arise in such a context. The conceptual relevance of the notion of an adjunction, which pervades category theory and, in a sense, Mathematics as a whole, justifies a somewhat more detailed introduction here. We will then revisit products and sums and introduce function spaces from the categorical point of view.

We have said in §5 that, by an entity being universal among a collection of similar ones, it is understood that there exists a unique way in which every other entity in the collection can be reduced to (or factored through) it. It turns out that a very general way to express such collections is as families of arrows, said *from* T *to* Z,

$$\{f : \mathsf{T}\ X \longrightarrow Z |\ X \in \mathrm{obj}(\mathsf{C})\}$$

for T a functor from C to D and Z an object of D.

An arrow ϵ_Z in such a family is *universal* if there is, for every $f : \mathsf{T}\ X \longrightarrow Z$, a unique arrow $f^\bullet : X \longrightarrow \mathsf{S}\ Z$ in C, such that f factors through ϵ_Z as expressed by the commutativity of the following diagram:

$$\begin{tikzcd}
X \arrow[d, "f^{\bullet}"'] & & \mathsf{T}\ X \arrow[d, "\mathsf{T}\ f^{\bullet}"'] \arrow[dr, "f"] & \\
\mathsf{S}\ Z & & \mathsf{T}\ \mathsf{S}\ Z \arrow[r, "\epsilon_Z"'] & Z
\end{tikzcd}$$

i.e., $f = \epsilon_Z \cdot \mathsf{T}\ f^{\bullet}$.

For the moment, think of $\mathsf{S}\ Z$ just as an object of C, related to Z in a way that will become clear soon. This object, as well as ϵ_Z, is of course unique (up to isomorphism). Finally, notice how the intuitive idea of *reducing* is formally captured in the notion of *factorisation*.

25. Product Revisited. Let us instantiate this definition with a familiar example. Take T as $\triangle$: $\mathsf{C} \longrightarrow \mathsf{C} \times \mathsf{C}$, the *diagonal* functor from C to its product category. Let $Z = \langle Z_1, Z_2\rangle$ be an object of $\mathsf{C} \times \mathsf{C}$. If it exists, an universal arrow from $\triangle$ to Z will be a pair $\epsilon_Z = \langle p_1, p_2\rangle$ of arrows such that, for each $\mathsf{C} \times \mathsf{C}$ arrow

$$f = \langle f_1, f_2\rangle :\triangle\ X \longrightarrow Z$$

there exists a unique $f^{\bullet}$ such that

$$\langle f_1, f_2\rangle = \langle p_1, p_2\rangle\cdot\ \triangle\ \ f^{\bullet}$$

Diagrammatically,

$$\begin{tikzcd}
X \arrow[d, "f^{\bullet}"'] & & \triangle\ X \arrow[d, "\triangle\ f^{\bullet}"'] \arrow[dr, "f=\langle f_1,f_2\rangle"] & \\
\mathsf{S}\ Z & & \triangle\ \mathsf{S}\ Z \arrow[r, "\langle p_1,p_2\rangle"'] & Z
\end{tikzcd}$$

As composition on $\mathsf{C} \times \mathsf{C}$ is defined componentwise, this can be written as

$$f_1 = p_1 \cdot f^{\bullet} \quad \text{and} \quad f_2 = p_2 \cdot f^{\bullet}$$

It is then straightforward to recognise $\mathsf{S}\ Z$ as the product $Z_1 \times Z_2$, p_1 and p_2 as the associated projections π_1 and π_2, and $f^{\bullet}$ as the *split* of f_1 and f_2. We, therefore, conclude that the pair $\langle \pi_1, \pi_2\rangle$ is the universal arrow in the family $\{f\ :\triangle\ \ X \longrightarrow \langle Z_1, Z_2\rangle |\ X \in \mathsf{obj}(\mathsf{C})\}$ and, for any f the *split* $\langle f_1, f_2\rangle$ of its components is the induced unique arrow.

26. PRODUCT AS A FUNCTOR. If the construction mentioned in the previous paragraph can be repeated for every object in $\mathsf{C} \times \mathsf{C}$, we get along all the binary products on C. Moreover, S emerges as a correspondence between pairs of C-objects and their product, which can be made *functorial* as follows. Let $\langle h_1, h_2 \rangle : \langle A_1, A_2 \rangle \longrightarrow \langle B_1, B_2 \rangle$ be a morphism in $\mathsf{C} \times \mathsf{C}$. Then, define,

$$h_1 \times h_2 \;=\; (\langle h_1, h_2 \rangle \cdot \epsilon_{\langle A_1, A_2 \rangle})^{\bullet}$$

and simplify

$$\begin{array}{cl}
 & (\langle h_1, h_2 \rangle \cdot \epsilon_{\langle A_1, A_2 \rangle})^{\bullet} \\
= & \qquad \{ \ \epsilon_{\langle A_1, A_2 \rangle} \text{ definition } \} \\
 & (\langle h_1, h_2 \rangle \cdot \langle \pi_1, \pi_2 \rangle)^{\bullet} \\
= & \qquad \{ \text{ composition in } \mathsf{C} \times \mathsf{C} \ \} \\
 & (\langle h_1 \cdot \pi_1, h_2 \cdot \pi_2 \rangle)^{\bullet} \\
= & \qquad \{ \ f^{\bullet} \text{ definition } \} \\
 & \langle h_1 \cdot \pi_1, h_2 \cdot \pi_2 \rangle
\end{array}$$

Back into the general case, note that in §24 we have been rather vague about construction $\mathsf{S}\ Z$. Provided universal arrows ϵ can be defined for each object of C, the answer is now obvious: S is a functor from D to C, whose action $\mathsf{S}\ h$, on a D-morphism $h : A \longrightarrow B$, is defined by the following diagram,

$$\begin{tikzcd}
\mathsf{S}\ A \arrow[d, "\mathsf{S}\ h = (h \cdot \epsilon_A)^{\bullet}"'] & & \mathsf{TS}\ A \arrow[d, "\mathsf{T}\ (h \cdot \epsilon_A)^{\bullet}"'] \arrow[dr, "h \cdot \epsilon_A"] & \\
\mathsf{S}\ B & & \mathsf{T}\,\mathsf{S}\ B \arrow[r, "\epsilon_B"'] & B
\end{tikzcd}$$

But let us come back again to the example at hands. Once product has been found functorial, the kit of laws introduced in §22 is automatically extended with the functorial laws (2.11) and (2.12), as well as a derived result showing a product of two arrows being 'absorbed' by a *split* of other two:

$$(i \times j) \cdot \langle g, h \rangle \;=\; \langle i \cdot g, j \cdot h \rangle \tag{2.28}$$

27. UNIVERSAL BECOMES NATURAL. Another consequence of the existence of universal arrows ϵ_Z from T to Z, for *each* object $Z \in \mathrm{obj}(\mathsf{D})$, is the emergence of a natural transformation

$$\epsilon : \mathsf{TS} \Longrightarrow \mathsf{Id}$$

that is, for the product case,

$$\epsilon : \Delta \times \Longrightarrow \mathsf{Id}_{\mathsf{C}\times\mathsf{C}}$$

i.e.,

$$\pi_1 \cdot (f \times g) = f \cdot \pi_1 \quad \text{and} \quad \pi_g \cdot (f \times g) = g \cdot \pi_2 \tag{2.29}$$

Another natural transformation which happens to play a complementary role with respect to ϵ_Z, is defined by considering the family of C-arrows corresponding to identities on objects $\mathsf{T}\ X$:

$$\eta : \mathsf{Id} \Longrightarrow \mathsf{ST}$$

For products, we have

$$\eta : \mathsf{Id}_{\mathsf{C}} \Longrightarrow \times\ \Delta$$

whose component, for each object X of C, is

$$\eta_X \;=\; (\mathsf{id}_{\Delta\ X})^{\bullet} : X \longrightarrow \times\ \Delta\ \ X$$

28. THE DUAL PICTURE. Each η_X can also be described as an universal arrow in the family of arrows

$$\{g : X \longrightarrow \mathsf{S}\ Z |\ Z \in \mathrm{obj}(\mathsf{D})\}$$

for S a functor from D to C and X an object of C.

Being universal in such a family of arrows (said *from X to* S) means, again, the existence of a unique factorisation. I.e., for all $g : X \longrightarrow \mathsf{S}\ Z$, there is a unique arrow $g_{\bullet} : \mathsf{T}\ Y \longrightarrow X$ such that $g = \mathsf{S}\ g_{\bullet} \cdot \eta_X$. In a diagram,

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, "g"'] & \mathsf{ST}\ X \arrow[d, "\mathsf{S}\ g_{\bullet}"] & & \mathsf{T}\ X \arrow[d, "g_{\bullet}"] \\
& \mathsf{S}\ Z & & Z
\end{tikzcd}$$

$\mathsf{T}\ X$ can, as we have first done with $\mathsf{S}\ Z$ in §24, be thought of as a mere object of D depending on X. However, as η is defined for all objects of C, T can be made into a functor from C to D. In particular, for any $h : A \longrightarrow B$, $\mathsf{T}\ h$ is defined by the following diagram:

$$\begin{tikzcd}
A \arrow[r, "\eta_A"] \arrow[dr, "\eta_B \cdot h"'] & \mathsf{ST}\ A \arrow[d, "\mathsf{S}\ (\eta_B \cdot h)_\bullet"] & & \mathsf{T}\ A \arrow[d, "\mathsf{T}\ h = (\eta_B \cdot h)_\bullet"] \\
& \mathsf{ST}\ B & & \mathsf{T}\ B
\end{tikzcd}$$

Finally, note that, if, in the beginning of this paragraph, η has been defined in terms of the unique factorisations of identities under ϵ, the converse is also true. In fact, ϵ can be defined as a natural transformation from TS to $\mathsf{Id_D}$ whose components are given by $\epsilon_Z = (\mathsf{id}_{\mathsf{S}\ Z})_\bullet$. As noticed above, we end up with two 'twin' (inter-definable) natural transformations ϵ and η.

29. EXAMPLE. As an example, take S as the diagonal functor $\triangle$ and seek for universal arrows from $X = \langle X_1, X_2\rangle$ to $\triangle$. The definition reads as expected: there is an arrow $\eta_Y = \langle q_1, q_2\rangle$ such that, for each arrow $g = \langle g_1, g_2\rangle : X \longrightarrow_{\triangle} Z$ in $\mathsf{C} \times \mathsf{C}$, there is a unique $g_\bullet : \mathsf{T}\ X \longrightarrow Z$ such that $g =_{\triangle} g_\bullet \cdot \langle q_1, q_2\rangle$, *i.e.*,

$$g_1 = g_\bullet \cdot q_1 \quad \text{and} \quad g_2 = g_\bullet \cdot q_2$$

Clearly, $\mathsf{T}\ X$ is the *coproduct* $X_1 + X_2$ and η_X is the pair of injections $\langle \iota_1 : X_1 \longrightarrow X_1 + X_2, \iota_2 : X_2 \longrightarrow X_1 + X_2\rangle$. Furthermore, $\langle h_1, h_2\rangle_\bullet$ is, by uniqueness, the *either* of h_1 and h_2. As such a construction is possible for every object in $\mathsf{C} \times \mathsf{C}$, the coproduct construction becomes a functor. Given an arrow $\langle h_1, h_2\rangle : \langle A_1, A_2\rangle \longrightarrow \langle B_1, B_2\rangle$ define

$$h_1 + h_2 \ = \ (\eta_{\langle B_1, B_2\rangle} \cdot \langle h_1, h_2\rangle)_\bullet = \ [\iota_1 \cdot h_1, \iota_2 \cdot h_2]$$

as expected, which paves the way for the dual 'absorption' law for sums:

$$[g, h] \cdot (i + j) \ = [g \cdot i, h \cdot j] \tag{2.30}$$

30. ADJUNCTIONS. As we have just seen, the existence of an universal arrow from a functor $\mathsf{T} : \mathsf{C} \longrightarrow \mathsf{D}$ to every D-object Z defines uniquely (up to isomorphism, of course!) a new functor $\mathsf{S} : \mathsf{D} \longrightarrow \mathsf{C}$. This is called the *right adjoint* of T. Similarly, if there exists an universal arrow from every C-object X to functor S, a functor T is uniquely defined. T is then called the *left adjoint* of S. This kind of relation between S and T is known as an *adjunction*. Adjoint functors are written $\mathsf{T} \dashv \mathsf{S}$.

As expected, there are two dual ways of defining an adjunction. In fact, the underlying symmetry in this notion can be made explicit by observing that η is obtained by reducing identities under T to ϵ and, similarly, ϵ results from the reduction of identities under S to η. Formally,

31. DEFINITION. A functor $\mathsf{T} : \mathsf{C} \longrightarrow \mathsf{D}$ is *left adjoint* to another functor $\mathsf{S} : \mathsf{D} \longrightarrow \mathsf{C}$, written

$$\mathsf{T} \dashv \mathsf{S}$$

if there is a natural transformation $\epsilon : \mathsf{TS} \Longrightarrow \mathsf{Id_D}$ such that, for all X in C and Z, $f : \mathsf{T}\ X \longrightarrow Z$ in D, there exists a unique arrow $g : X \longrightarrow \mathsf{S}\ Z$ such that $f = \epsilon_Z \cdot \mathsf{T}\ g$. Usually, g is written as $f^\bullet$ in order to emphasise its uniqueness upon f.

Alternatively, if there is a natural transformation $\eta : \mathsf{Id_C} \Longrightarrow \mathsf{ST}$ such that, for all X and $g : X \longrightarrow \mathsf{S}\ Z$ in C and Z in D, there exists a unique arrow $f : \mathsf{T}\ X \longrightarrow Z$ such that $g = \mathsf{S}\ f \cdot \eta_X$. Dually, f is written as $g_\bullet$.

In both cases, these (equivalent) definitions guarantee the existence of 'enough' universals. In consequence, an adjunction gives rise to a (natural) bijective correspondence between arrows $f : \mathsf{T}\ X \longrightarrow Z$, in D, and $g : X \longrightarrow \mathsf{S}\ Z$, in C, captured by the following equivalence

$$g = \mathsf{S}\ f \cdot \eta \quad \Leftrightarrow \quad f = \epsilon \cdot \mathsf{T}\ g \tag{2.31}$$

Yet another popular definition of $\mathsf{T} \dashv \mathsf{S}$ is formulated simply in terms of the following conditions (known as the 'triangle equalities') on η and ϵ:

$$\mathsf{S}\ \epsilon \cdot \eta\mathsf{T}\ = 1_\mathsf{S} \tag{2.32}$$

$$\epsilon\mathsf{T} \cdot \mathsf{T}\ \eta\ = 1_\mathsf{T} \tag{2.33}$$

In any case, η (resp. ϵ) is called the *unit* (resp. *counit*) of the adjunction.

32. LIMITS AND COLIMITS. In §§25, 29 we have discussed, in some detail, how products and coproducts arise as, respectively, left and right adjoints to the diagonal functor. In fact, adjunctions $+ \dashv \vartriangle$ and $\vartriangle \dashv \times$ are particular cases of two fundamental families of adjunctions: the ones that give rise to limits and colimits in general. A basic observation is the isomorphism between $\mathsf{D} \times \mathsf{D}$ and the functor category $\mathsf{D}^\mathbf{2}$, where $\mathbf{2}$ is the category with two objects and no arrows other than the associated identities. We may now generalise the notion of a diagonal functor and look for its right and left adjoints. First the generalized diagonal functor $\vartriangle_\mathsf{H}$: $\mathsf{C} \longrightarrow \mathsf{C}^\mathsf{H}$ is defined, for H is any small category. Clearly $\vartriangle_\mathbf{2}$ is the usual $\vartriangle$.

Next, take $\vartriangle_\mathbf{1}$: $\mathsf{C} \longrightarrow \mathsf{C}^\mathbf{1}$, which maps every object in C into the unique functor from $\mathbf{1}$ to C. Its right adjoint, if it exists at all, maps this unique object of $\mathsf{C}^\mathbf{1}$ into the final object of C and the corresponding η_X, for all $X \in \mathrm{obj}(\mathsf{C})$, coincides with $!_X$. Dually, a left adjoint would give the initial object and identify ϵ_X with $?_X$.

We can think of functors from H to C as H-shaped *diagrams* in C. In general, right (resp. left) adjoints to $\vartriangle_\mathsf{H}$ give the limit (resp. colimit) of such diagrams. For example, taking H as the category with three objects depicted as follows

$$\begin{tikzcd} \bullet \arrow[r] & \bullet & \bullet \arrow[l] \end{tikzcd}$$

the right adjoint to Δ_{H} defines *pullbacks*. Similarly, *pushouts* arise by reversing the arrows in H above and taking the left adjoint to the same 'diagonal' functor.

A basic result on adjunctions states that, in an adjunction situation $\mathsf{T} \dashv \mathsf{S}$, the left adjoint, T, preserves all colimits while, dually, the right adjoint S does the same for limits.

33. EXPONENTIALS. The categorical version of the usual notion of a function space in Set also arises, as one could expect, from an adjunction situation. The rather heavy use in this thesis of its laws justifies a detailed introduction here.

Let C be an object of C and suppose that $\mathsf{Id} \times C$, the C-section of the product bifunctor, has a right adjoint which we shall denote by Id^C. This means that for all $f : X \times C \longrightarrow Y$, there exists a unique $f^\bullet : X \longrightarrow Y^C$ such that $f = \epsilon_Y \cdot (f^\bullet \times C)$, both the object Y^C and the universal ϵ_Y being uniquely determined up to isomorphism. Diagrammatically,

$$\begin{tikzcd}
X \arrow[d, "f^\bullet"'] \\
Y^C
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
X \times C \arrow[d, "f^\bullet \times \mathsf{id}_C"'] \arrow[dr, "f"] & \\
Y^C \times C \arrow[r, "\epsilon_Y"'] & Y
\end{tikzcd}$$

Following the general construction of §26, Id^C extends to a functor by defining, for $h : A \longrightarrow B$,

$$h^C : A^C \longrightarrow B^C \;=\; (h \cdot \epsilon_A)^\bullet \tag{2.34}$$

Are we done? In fact, note that Y^C has exactly the characteristic properties of the set of functions from C to Y in Set. Bijection $f \leftrightsquigarrow f^\bullet$ corresponds, in this particular context, to *currying*: the well-known isomorphism between (binary) functions from $X \times C$ to Y and (unary) functions from X to the set of functions from C to Y. Being so popular, this terminology is also adopted in an arbitrary category: $f^\bullet$ is called the *curry* of f and is written $\overline{f}$.

The family $\epsilon_X : X^C \times C \longrightarrow X$ is, of course, the counit of adjunction

$$\mathsf{Id} \times C \;\dashv\; \mathsf{Id}^C$$

On the other hand, its unit has $\eta_X : X \longrightarrow (X \times C)^C$ as components. In Set, ϵ corresponds to function evaluation and η to a function constructor:

$$\begin{aligned}
\epsilon\ \langle f, c\rangle &= f\ c \\
\eta\ x &= \lambda c.\ \langle x, c\rangle
\end{aligned}$$

Counit ϵ is more commonly named ev, after *evaluation*. We shall refer to η as sp, after *stamping*, and, again, such designations will carry over to general case.

34. EXPONENTIAL LAWS. Adjunction $\mathsf{Id} \times C \dashv \mathsf{Id}^C$ entails an universal characterisation of exponentials:

$$k = \overline{f} \;\Leftrightarrow\; f = \mathsf{ev} \cdot (k \times \mathsf{id}) \tag{2.35}$$

from which the following laws are derived

$$f = \mathsf{ev} \cdot (\overline{f} \times \mathsf{id}) \tag{2.36}$$

$$\overline{\mathsf{ev}} = \mathsf{id}_{X^C} \tag{2.37}$$

$$\mathsf{sp} = \overline{\mathsf{id}_{X \times C}} \tag{2.38}$$

$$\overline{g} \cdot f = \overline{g \cdot (f \times \mathsf{id})} \tag{2.39}$$

In an arbitrary category with exponentials C, A^C represents, as a C-object, the arrows from C to A. Consequently, the action of Id^C on each arrow $f : A \longrightarrow B$ should *internalise* composition with f. In Set it is easy to verify that this is indeed the case. For $g : C \longrightarrow A$ and $c \in C$, a simple calculation yields,

$$\begin{array}{cl}
 & (f^C\ g)\ c \\
= & \quad \{ \ \mathsf{Id}^C \text{ on arrows (2.34)} \ \} \\
 & (\overline{(f \cdot \mathsf{ev})}\ g)\ c \\
= & \quad \{ \ \text{uncurrying} \ \} \\
 & f \cdot \mathsf{ev}\ \langle g, c \rangle \\
= & \quad \{ \ \text{function composition} \ \} \\
 & f\ (\mathsf{ev}\ \langle g, c \rangle) \\
= & \quad \{ \ \mathsf{ev} \text{ defi nition} \} \\
 & f\ (g\ c) \\
= & \quad \{ \ \text{function composition} \ \} \\
 & (f \cdot g)\ c
\end{array}$$

In an arbitrary category, however, we cannot talk about ‘applying’ a morphism to an ‘element’ of an object. We have, then, to state and prove the intended result in the language of generalized elements (§7). A generalized element of an exponential A^C is an arrow $\overline{g} : T \longrightarrow A^C$, which corresponds uniquely, under the adjunction, to $g : T \times C \longrightarrow A$. Keeping in mind that, in the generalized elements notation, $f^C\ \overline{g}$

corresponds to $f^C \cdot \overline{g}$, the 'internalisation' result takes the form of an 'absorption' property for exponentials:

$$\overline{f \cdot g} \;= f^C \cdot \overline{g} \tag{2.40}$$

Proof. Consider the following diagram

$$\begin{tikzcd}
T \times C \arrow[r, "\overline{g}\times C"] \arrow[dr, "g"'] & A^C \times C \arrow[r, "f^C \times C"] \arrow[d, "\mathsf{ev}_A"] & B^C \times C \arrow[d, "\mathsf{ev}_B"] \\
& A \arrow[r, "f"'] & B
\end{tikzcd}$$

and note that the left triangle commutes by definition of $\overline{g}$ and the square because ev is natural. Therefore,

$$\begin{array}{rl}
& f \cdot g \;=\; \mathsf{ev}_B \cdot (f^C \times C) \cdot (\overline{g} \times C) \\
\equiv & \qquad \{\ \times \text{ functor} \ \} \\
& f \cdot g \;=\; \mathsf{ev}_B \cdot (f^C \cdot \overline{g} \times C) \\
\equiv & \qquad \{\ \text{exponential universal property (2.35)} \ \} \\
& \overline{f \cdot g} \;=\; f^C \cdot \overline{g}
\end{array}$$

□

Notice that the pointwise calculation above can be rephrased, using this result, and taking g as a *point*, *i.e.*, $\overline{g} : \mathbf{1} \longrightarrow A^C$. In this case, $f^C \ \overline{g}$ equals $\overline{f \cdot g}$ as proved above, but now $\overline{f \cdot g}$ is itself a point of B^C, which corresponds to morphism $f \cdot g$. In other words,

$$f^C \;=\; f \cdot \_$$

35. THE EXPONENTIAL BIFUNCTOR. The exponential functor above can be made into a bifunctor by defining, for each $h : C \longrightarrow D$, an arrow $X^h : X^D \longrightarrow X^C$ as follows:

$$X^h \;\triangleq\; X^D \xrightarrow{\mathsf{sp}} (X^D \times C)^C \xrightarrow{(\mathsf{id}_{X^D} \times h)^C} (X^D \times D)^C \xrightarrow{\mathsf{ev}^C} X^C$$

Note that the exponential bifunctor is *contravariant* in its second argument. Moreover, X^h can be proved equal to post-composition with h, *i.e.*, $X^h = \_ \cdot h$.

36. CARTESIAN CATEGORIES. Categories are classified according to the structure they exhibit. A category with all finite products, or, equivalently, with binary products and final object, is called *Cartesian*.

Observe that the product construction on a category has the structure (up to isomorphism) of an Abelian monoid. To establish notation, let us represent associativity, commutativity and right and left units by the following isomorphisms, natural on A, B and C:

$$\begin{aligned}
\mathsf{a} &: A \times B \times C \longrightarrow A \times (B \times C) \\
\mathsf{s} &: A \times B \longrightarrow B \times A \\
\mathsf{r} &: \mathbf{1} \times A \longrightarrow A \\
\mathsf{l} &: A \times \mathbf{1} \longrightarrow A
\end{aligned}$$

whose inverses are, respectively, a°, s (notice s is its own inverse), r° and l°. In any Cartesian category they are defined as follows:

$$\begin{aligned}
\mathsf{a} &= \langle \pi_1 \cdot \pi_1, \langle \pi_1 \cdot \pi_2, \pi_2 \rangle \rangle \\
\mathsf{s} &= \langle \pi_2, \pi_1 \rangle \\
\mathsf{r}^\circ &= \langle !_A, \mathsf{id}_A \rangle \\
\mathsf{l}^\circ &= \langle \mathsf{id}_A, !_A \rangle
\end{aligned}$$

37. HOUSEKEEPING MORPHISMS. The following morphisms provide a shorthand notation for typical combinations of a and s. We call them *exchange* morphisms as they change the position of some factors in a multiplicative expression. They are particularly useful to handle 'housekeeping' tasks when calculating in a cartesian category.

$$\begin{aligned}
\mathsf{xr} &: A \times B \times C \longrightarrow A \times C \times B \\
&= \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a} \\
\mathsf{xl} &: A \times (B \times C) \longrightarrow B \times (A \times C) \\
&= \mathsf{a} \cdot (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\
\mathsf{m} &: A \times B \times (C \times D) \longrightarrow A \times C \times (B \times D) \\
&= \mathsf{a} \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{a}^\circ \;=\; \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{xl}) \cdot \mathsf{a}
\end{aligned}$$

If the category has finite coproducts, we shall refer to the corresponding associativity, commutativity and unit morphisms as

$$\begin{aligned}
\mathsf{a}_+ &: A+B+C \longrightarrow A+(B+C)\\
\mathsf{s}_+ &: A+B \longrightarrow B+A\\
\mathsf{r}_+ &: \emptyset + A \longrightarrow A\\
\mathsf{l}_+ &: A+\emptyset \longrightarrow A
\end{aligned}$$

and consider the additive versions of the 'exchange' morphisms:

$$\begin{aligned}
\mathsf{xr}_+ &: A+B+C \longrightarrow A+C+B\\
\mathsf{xl}_+ &: A+(B+C) \longrightarrow B+(A+C)\\
\mathsf{m}_+ &: (A+B)+(C+D) \longrightarrow (A+C)+(B+D)
\end{aligned}$$

38. CARTESIAN CLOSEDNESS. A Cartesian category in which product has a right adjoint is classified as *Cartesian closed* (or *ccc*, for short). Such a category has exponentials, and therefore the capacity of representing hom-sets as objects and internalising composition. Set is a prime example.

39. DISTRIBUTIVITY. Suppose a category has both finite products and coproducts. If, additionally, binary products distribute over finite coproducts, the category is called *distributive*. Being distributive means there exist two natural isomorphisms

$$\begin{aligned}
\mathsf{dl} &: (A+B)\times C \longrightarrow (A\times C)+(B\times C)\\
\mathsf{zl} &: \emptyset \times A \longrightarrow \emptyset
\end{aligned}$$

as finite coproducts are generated from binary and nullary coproducts. The latter is, of course, the initial object in the category. 'Right' versions of these isomorphisms are obtained by pre-composing them with s. We shall denote them as $\mathsf{dr} : C\times(A+B) \longrightarrow (C\times A)+(C\times B)$ and $\mathsf{zr} : A\times\emptyset \longrightarrow \emptyset$. Set, as any other *ccc* with coproducts, is distributive. On the other hand, Rel is not. Although distributivity is a much weaker notion than, for instance, cartesian closedness, it has been proposed, notably by R. C. Walters [Wal89], as 'the' natural semantic framework for datatypes and programming. It is also the basic requirement on the semantic category underlying CHARITY (see appendix E). References [Coc93] and [CLW93] provide details on this topic.

Let us make a brief incursion on distributivity. First notice that the inverse of dl can be defined easily as

$$\mathsf{dl}^\circ = [\iota_1 \times C, \iota_2 \times C]$$

or, by application of the exchange law (2.27), as a *split* involving the same arrows. On the other hand, dl has no pointfree definition in terms of *eithers* or *splits* alone. However, if the category is ccc, it can be defined pointfree as follows:

$$(A+B)\times C \xrightarrow{[\iota_1^C\cdot\mathsf{sp},\iota_2^C\cdot\mathsf{sp}]\times C} (A\times C+B\times C)^C\times C \xrightarrow{\mathsf{ev}} A\times C+B\times C$$

Let us check the correctness of this definition. This involves two facts $\mathsf{dl}^\circ\cdot\mathsf{dl}=\mathsf{id}$ and $\mathsf{dl}\cdot\mathsf{dl}^\circ=\mathsf{id}$ which will be proved as an exercise in using the categorical 'tool kit' introduced along this chapter.

Proof. First verify

$$\begin{array}{ll}
 & \mathsf{dl}^\circ\cdot\mathsf{dl} \\
= & \quad\{\ \mathsf{dl}\text{ definition}\ \} \\
 & \mathsf{dl}^\circ\cdot\mathsf{ev}\cdot([\iota_1^C\cdot\mathsf{sp},\iota_2^C\cdot\mathsf{sp}]\times C) \\
= & \quad\{\ \mathsf{ev}\text{ natural}\ \} \\
 & \mathsf{ev}\cdot(\mathsf{dl}^{\circ C}\times C)\cdot([\iota_1^C\cdot\mathsf{sp},\iota_2^C\cdot\mathsf{sp}]\times C) \\
= & \quad\{\ \times\text{ functor}\ \} \\
 & \mathsf{ev}\cdot(\mathsf{dl}^{\circ C}\cdot[\iota_1^C\cdot\mathsf{sp},\iota_2^C\cdot\mathsf{sp}]\times C) \\
= & \quad\{\ +\text{-fusion, } \textit{exponential}\text{ functor}\ \} \\
 & \mathsf{ev}\cdot[(\mathsf{dl}^\circ\cdot\iota_1)^C\cdot\mathsf{sp},(\mathsf{dl}^\circ\cdot\iota_2)^C\cdot\mathsf{sp}]\times C \\
= & \quad\{\ \mathsf{dl}^\circ\text{ definition, }+\text{-cancellation}\ \} \\
 & \mathsf{ev}\cdot[(\iota_1\times C)^C\cdot\mathsf{sp},\iota_2\times C)^C\cdot\mathsf{sp}]\times C \\
= & \quad\{\ \mathsf{sp}\text{ natural}\ \} \\
 & \mathsf{ev}\cdot[\mathsf{sp}\cdot\iota_1,\mathsf{sp}\cdot\iota_2]\times C \\
= & \quad\{\ +\text{ fusion}\ \} \\
 & \mathsf{ev}\cdot(\mathsf{sp}\cdot[\iota_1,\iota_2]\times C) \\
= & \quad\{\ +\text{ reflection}\ \} \\
 & \mathsf{ev}\cdot(\mathsf{sp}\times C) \\
= & \quad\{\ \text{adjunction (2.31)}\ \} \\
 & \mathsf{id}_{(A+B)\times C}
\end{array}$$

and then

$$\mathsf{dl}\cdot\mathsf{dl}^\circ$$

$$
\begin{array}{ll}
= & \{\ \mathsf{dl} \text{ and } \mathsf{dl}^\circ \text{ definition}\} \\
 & \mathsf{ev} \cdot ([\iota_1^C \cdot \mathsf{sp}, \iota_2^C \cdot \mathsf{sp}] \times C) \cdot [\iota_1 \times C, \iota_2 \times C] \\
= & \{\ + \text{ fusion }\} \\
 & \mathsf{ev} \cdot [([\iota_1^C \cdot \mathsf{sp}, \iota_2^C \cdot \mathsf{sp}] \times C) \cdot (\iota_1 \times C), ([\iota_1^C \cdot \mathsf{sp}, \iota_2^C \cdot \mathsf{sp}] \times C) \cdot (\iota_2 \times C)] \\
= & \{\ \times \text{ functor and } + \text{ cancellation }\} \\
 & \mathsf{ev} \cdot [(\iota_1^C \cdot \mathsf{sp}) \times C, (\iota_2^C \cdot \mathsf{sp}) \times C] \\
= & \{\ + \text{ fusion }\} \\
 & [\mathsf{ev} \cdot ((\iota_1^C \cdot \mathsf{sp}) \times C), \mathsf{ev} \cdot ((\iota_2^C \cdot \mathsf{sp}) \times C)] \\
= & \{\ \times \text{ functor}\} \\
 & [\mathsf{ev} \cdot (\iota_1^C \times C) \cdot (\mathsf{sp} \times C), \mathsf{ev} \cdot (\iota_2^C \times C) \cdot (\mathsf{sp} \times C)] \\
= & \{\ \mathsf{ev} \text{ natural and (2.31) }\} \\
 & [\iota_1, \iota_2] \\
= & \{\ + \text{ reflection }\} \\
 & \mathsf{id}_{(A \times C)+(B \times C)}
\end{array}
$$

□

40. CONDITIONALS. In a distributive category conditional expressions can be modelled by coproducts. In this thesis we adopt the McCarthy conditional constructor written as $(p \to f, g)$, where $p : A \longrightarrow \mathbf{2}$ is a predicate. Intuitively, $(p \to f, g)$ reduces to f if p evaluates to `true` and to g otherwise. The conditional construct is defined as

$$(p \to f, g) \;=\; \langle f, g \rangle \cdot p?$$

where $p? : A \longrightarrow A + A$ is determined by predicate p as follows

$$p? = \qquad A \xrightarrow{[\mathsf{id},p]} A \times (\mathbf{1}+\mathbf{1}) \xrightarrow{\mathsf{dl}} A \times \mathbf{1} + A \times \mathbf{1} \xrightarrow{\pi_1+\pi_1} A + A$$

Reference [Gib97] provides a comprehensive set of laws to calculate with conditionals. The following will be used in this thesis:

$$h \cdot (p \to f, g) \;=\; (p \to h \cdot f, h \cdot g) \tag{2.41}$$

$$(p \to f, g) \cdot h \;=\; (p \cdot h \to f \cdot h, g \cdot h) \tag{2.42}$$

$$(p \to f, g) \;=\; (p \to (p \to f, g), (p \to f, g)) \tag{2.43}$$

CHAPTER 3

Algebras, Coalgebras and Categorical Data Types

Summary

One way of thinking about a coalgebra is as a transition structure with a particular shape encoding the ways in which a state space is accessed through observers and actions. The semantics of such a structure is observational, in the sense that its internal configurations remain hidden and have, therefore, to be identified, if not distinguishable by observation. This chapter provides some intuition on the notions of a coalgebra and its dual, and presents some background material needed along the thesis. In particular, algebras and coalgebras for an endofunctor, and the related concepts of comorphism and bisimulation, are reviewed in some detail. This leads to an incursion on categorical data types, as a framework in which both data structures and behavioural patterns can be abstracted.

1. Observation And Construction

1. FUNCTIONS. One of the most elementary models of a computational process is that of a *function*

$$f : I \longrightarrow O$$

which specifies a transformation rule between two structures I and O. The behaviour of a function is captured by the output it produces, which is completely determined by the supplied input. In a (metaphorical) sense, this may be dubbed as the 'engineer's view' of reality: *here is a recipe (a tool, a technology) to build gnus from gnats*.

Often, however, reality is not so simple. For example, one may know how to produce 'gnus' from 'gnats' but not in all cases. This is expressed by observing the output of f in a more refined context: O is replaced by $O + \mathbf{1}$ and f is said to be a *partial* function. In other situations one may recognise that there is some environmental (or context) information about 'gnats' that, for some reason, should be hidden from input. It may be the case that such information is too extensive to be supplied to f by its user, or that it is shared by other functions as well. It might also be the case that building gnus would eventually modify the environment, thus influencing latter production of more 'gnus'. For U a denotation of such context information, the signature of f becomes

$$f : I \longrightarrow (O \times U)^U$$

In both cases f can be typed as

$$f : I \longrightarrow \mathsf{T}\, O$$

for $\mathsf{T} = \mathsf{Id} + \mathbf{1}$ and $\mathsf{T} = (\mathsf{Id} \times U)^U$, respectively. Informally, T can be thought of as a type transformer providing a *shape* for the output of f. Technically, T is a *functor* (§2.14) which, to facilitate composition and manipulation of such functions, is often required to be a *monad* (§A.2). In this way, the 'universe' in which $f : I \longrightarrow \mathsf{T}\ O$ lives and is reasoned about is the *Kleisli category* for T (§A.7). In fact, monads in functional programming offer a general technique to smoothly incorporate, and delimit, 'computational effects' of this kind without compromising the purely functional semantics of such languages, in particular, referential transparency.

2. STATE. A function computed within a context is often referred to as 'state-based', in the sense the word 'state' has in automaton theory — the internal memory of the automaton which both constraints and is constrained by the execution of actions. In fact, the 'nature' of $f : I \longrightarrow (O \times U)^U$ as a 'state-based function' is made more explicit by rewriting its signature as

$$f : U \longrightarrow (O \times U)^I$$

This, in turn, may suggest an alternative model for computations, which (again in a metaphorical sense) one may dub as the 'natural scientist's view'. Instead of a recipe to build 'gnus' from 'gnats', the simple awareness that *there exist gnus and gnats and that their evolution can be observed*. That *observation* may entail some form of *interference* is well known, even from Physics, and thus the underlying notion of computation is not necessarily a passive one.

The able 'natural scientist' will equip herself with the right 'lens' — that is, a tool to observe with, which necessarily entails a particular shape for observation. Also note that the emphasis is now placed on the state itself: the input and output parameters may or may not become relevant, depending on the particular kind of observation one may want to perform. In other words, one's focus becomes the 'nature', or the 'universe' or, more pragmatically, the *state space*. That we can observe 'gnus' being produced out of 'gnats' is just one, among other possible observations. The basic ingredients required to support an 'observational', or 'state-based', view of computational processes may be summarised as follows:

a *lens*:	○⌒○	a functor T
an *observation structure*:	universe $\xrightarrow{p}$ ○⌒○ universe	a T-coalgebra

Formally, in Set, a *coalgebra* for a functor T is a set U, which corresponds to the object being observed (the 'universe'), and a function $p : U \longrightarrow \mathsf{T}\ U$. Such a function is often referred to as the coalgebra *dynamics*.

3. COLOURS. There is, of course, a great diversity of 'lenses' and, for the same 'lens', a variety of observation structures, *i.e.*, of coalgebras. Moreover, such structures can be related and compared. For this one only needs what in Universal Algebra is known as a *homomorphism*, *i.e.*, a structure-preserving map. In our case the structure to be preserved is the shape of T as an observation tool. Therefore, a T-coalgebra morphism (or *comorphism*, as abbreviated in the sequel) h between, say, coalgebras p and q is just a function between the respective carriers ('universes' or 'state-spaces') making the following diagram to commute:

$$\begin{tikzcd}
U \arrow[r, "p"] \arrow[d, "h"'] & \mathsf{T}\ U \arrow[d, "\mathsf{T}\ h"] \\
V \arrow[r, "q"'] & \mathsf{T}\ V
\end{tikzcd}$$

Let us consider some possible lenses. An extreme case is the opaque lens: no matter what we try to observe through it, the outcome is always the same. Formally, such a lens is the constant functor $\underline{\mathbf{1}}$ (§2.16) which maps every object to $\mathbf{1}$ and every morphism to the identity on $\mathbf{1}$. Since $\mathbf{1}$ is the final object (§2.6) in Set, all $\mathbf{1}$-coalgebras reduce to $!$.

A slightly more interesting lens is $\underline{\mathbf{2}}$, which allows states to be classified into two different classes: black or white. This makes it possible to identify *subsets* of the 'universe' U under observation, as an observation structure p for this functor will map elements of U to one or another element of $\mathbf{2}$.

Should an arbitrary set O be chosen to colour one's lens, the possible observations become more discriminating. A coalgebra for $\underline{O}$ is a 'colouring' device in the sense that elements of the universe are classified (*i.e.*, regarded as distinct) by being assigned to different elements of O. Of course, a map h between such two observation structures p and q should be a colour-preserving function, *i.e.*, equation

$$q \cdot h \;=\; p$$

must hold. This means that if two elements of the universe are grouped together by p, their images under h remain together when compared by q.

4. INTERFACES. A 'colour set' as $\mathbf{1}$, $\mathbf{2}$ or O above, can be regarded as a *classifier* of the state space. Coalgebras, for such constant functors, are *pure* observers providing a limited access to the state space by mapping into the 'colour set'. In object-oriented programming they are known as *attributes*. Naturally, the same 'universe' can be observed through different attributes and, furthermore, such observations can be carried out on parallel. Thus, the shape of a 'multi-attribute' lens is

$$\bigcirc\!\frown\!\bigcirc\; U \;=\; \prod_{k \in K} O_k$$

where K is a finite set of attribute names. The corresponding observation structure, a function mapping U to a (finite) product, is defined as a K-indexed *split*

$$\langle o_k \rangle_{k \in K} : U \longrightarrow \prod_{k \in K} O_k$$

of attributes o_k from U to O_k.

A very common assumption in state-based computations is that the state itself is a 'black box': it may evolve either internally or as a reaction to external stimuli, but the only way one has to become aware of such an evolution is by observing the values of its attributes. The product of their types forms the *output interface* of the coalgebra. No direct access to the state space is possible. Under this assumption the 'transparent' lens is not particularly useful. Technically, this lens corresponds to the *identity* functor Id. An observation structure for Id amounts to a function

$$p : U \longrightarrow U$$

This means that, by using p, the state U can indeed be modified, an ability we hadn't before. But, on the other hand, the absence of attributes makes any meaningful observation impossible. The best we can say, if no direct access to U is allowed, is just that *things happen*.

A better alternative is to combine attributes with such state modifiers, or update operations, to model the 'universe' evolution. The latter will be called *actions* here; in the object paradigm they are known as *methods*. Such a combination leads to a richer stock of lens. We might consider, for example, that

(1) *things happen and disappear* or *stop*, *i.e.*

$$\bigcirc\!\frown\!\bigcirc\, U \;=\; U + \mathbf{1}$$

The observation structures for this functor are the partial functions. Accordingly, morphisms between them consist of functions that preserve partiality.

(2) *things happen and, in doing so, some of their attributes become visible*, *i.e.*, (non trivial) output is produced:

$$\bigcirc\!\frown\!\bigcirc\, U \;=\; U \times O$$

(3) *the evolution of things is triggered by some external stimulus*, *i.e.*, additional *input* is accepted:

$$\bigcirc\!\frown\!\bigcirc\, U \;=\; U^{I}$$

(4) *we are not completely sure about what has happened*, in the sense that the evolution of the system being observed may be nondeterministic. In this case, the lens above can be combined with

$$\bigcirc\!\frown\!\bigcirc\, U \;=\; \mathcal{P}U$$

where $\mathcal{P}U$ is the finite powerset of U.

In the third example, the action also has an *input interface*. Typically, actions over the same state space cannot happen simultaneously and, therefore, if more than one is specified in a particular structure, in each execution the input supplied will also select the action to be activated. In some cases, the input is there only for selection purposes: actions with trivial input (*i.e.*, $I = \mathbf{1}$) correspond to buttons that can be pressed. Then the input interface organises itself as a coproduct. A possible shape for a sophisticated lens with both attributes and actions is

$$\bigcirc\!\frown\!\bigcirc\, U \;=\; \prod_{k \in K} O_k \times U^{\sum_{j \in J} I_j}$$

whose coalgebras are

$$\langle \langle o_k \rangle_{k \in K}, menu \rangle : U \longrightarrow \bigcirc\!\frown\!\bigcirc\, U$$

where $menu$ can be specified as

$$menu \;=\; \overline{[a_j]_{j \in J} \cdot \mathsf{dr}_J}$$

where dr_J is the suitable distribution law and each $a_j : U \times I_j \longrightarrow U$ is an elementary action. There are, of course, other alternatives to combine actions and attributes into sophisticated observation structures.

5. Combining Lenses. The tree below depicts several combinations of three of the basic alternative lenses of §4 (nondeterministic observations are left aside for the moment):

$$\begin{tikzcd}
 & & (O \times \mathsf{Id})^I \arrow[r] & (O \times \mathsf{Id})^I + \mathbf{1} & \text{(a)} \\
 & O \times \mathsf{Id} \arrow[ur] \arrow[dr] & & & \\
 & & (O \times \mathsf{Id}) + \mathbf{1} \arrow[r] & ((O \times \mathsf{Id}) + \mathbf{1})^I & \text{(b)} \\
 & & O \times (\mathsf{Id} + \mathbf{1}) \arrow[r] & (O \times (\mathsf{Id} + \mathbf{1}))^I & \text{(c)} \\
\mathsf{Id} \arrow[r] \arrow[uuur] \arrow[dddr] & \mathsf{Id} + \mathbf{1} \arrow[ur] \arrow[dr] & & & \\
 & & (\mathsf{Id} + \mathbf{1})^I \arrow[r] & O \times (\mathsf{Id} + \mathbf{1})^I & \text{(d)} \\
 & & O \times \mathsf{Id}^I \arrow[r] & (O \times \mathsf{Id}^I) + \mathbf{1} & \text{(e)} \\
 & \mathsf{Id}^I \arrow[ur] \arrow[dr] & & & \\
 & & \mathsf{Id}^I + \mathbf{1} \arrow[r] & O \times (\mathsf{Id}^I + \mathbf{1}) & \text{(f)}
\end{tikzcd}$$

It is instructive to see what happens in each case if the state space ‘collapses’, *i.e.*, if the ‘universe’ becomes trivial or, formally, if U is identified with $\mathbf{1}$:

$$\begin{tikzcd}
\text{(a)} \arrow[d, dash, dotted] & \text{(b)} \arrow[d, dash, dotted] & \text{(c)} \arrow[d, dash, dotted] & \text{(d)} \arrow[d, dash, dotted] & \text{(e)} \arrow[d, dash, dotted] & \text{(f)} \arrow[d, dash, dotted] \\
O^I + \mathbf{1} & (O + \mathbf{1})^I & (O \times \mathbf{2})^I & O \times \mathbf{2}^I & O + \mathbf{1} & O \times \mathbf{2}
\end{tikzcd}$$

We conclude that (b) as a set is isomorphic to the space of partial functions, (e) is just a pointed set and (a) a pointed function space. On the other hand, (d) is a set and a predicate and in (f) the structure has boiled down simply to a set and a Boolean flag.

6. INTERACTION. Another way of regarding observation structures is as *transition systems* over the state space. For example, coalgebras over both Id^I and $\mathsf{Id} \times O$ can be described in terms of *transitions* of the form

$$u \xrightarrow{x} u'$$

where $u, u' \in U$ and $x \in I$ or $x \in O$ in, respectively, the former and the latter case. Depending on how this transition relation is interpreted, we may classify the system as *reactive* or *active*, respectively.

- In the first case $u \xrightarrow{x} u'$ means that, for a coalgebra p, $(p\ u)\ x = u'$. Therefore p models a *reactive* system in which the transition can be read as *state u is able to accept x and move into state u'.*
- In the second case, the transition relation reads as *generate x in state u and then become u'*. Rather than a stimulus, x is an outcome and such a system is called *active*.

This distinction may not be intrinsic to the system being observed. It is essentially a distinction on the *shape* of observation. The lenses of the two examples complement each other. Recall that, similarly, finite automaton theory deals with automata *accepting* (*recognising*) or *generating* a particular language.

7. OBSERVATION EQUIVALENCE. Given a particular lens T and a T-observation structure p, when can two states be taken as 'equivalent'? If it is not possible to access their internal structure, all we can say is that they should be identified if all the observations that can be made over one and the other exhibit the same outcome and this remains true along the whole evolution of the system. Thinking of observation structures as transition systems, the notion of *bisimulation* [Par81, Mil80] can be recalled as precisely such a relation.

Whatever is observed of a system constitutes its *behaviour*, *i.e.*, the possible patterns of interaction with the observer. Two states are said bisimilar if they have (or *generate*, or *unfold to*) the same behaviour. Thus, equivalence means *indistinguishability* under observation. What coalgebra theory offers is a notion of *bisimulation* parametric on the particular 'lens' used. In other words, bisimilarity *acquires a shape*. An important fact is that, providing lens T is 'smooth enough', there exists a *canonical* representation embodying all T-behaviours into a T-coalgebra as well. Canonical in the sense that, from every other observation structure for the same lens, there is one and just one morphism to it. Technically, such a coalgebra, usually denoted by $\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\ \nu_{\mathsf{T}}$, is said to be the *final* coalgebra. The unique morphism mapping any other coalgebra p to it, which unfolds p to its behaviour, is represented by $[\!(p)\!]_{\mathsf{T}}$ and is called the *anamorphism* generated by p [MFP91], the *coinductive extension* of p [TR98] or, simply, the *final semantics* of the states of p [RT94].

8. Construction Strikes Back. On the other hand, and returning to our metaphor , the 'engineer's view' emphasises the possibility of at least some (essentially finite) things being not only observed, but actually *built*. In this case, one works not with a 'lens' but with a 'toolbox'. The *assembly process* is specified in a similar (but dual) way to the one used to define observation structures. I.e, the engineer will become equipped with,

a *tool box*:	🧰	a functor T
an *assembly process*:	🧰 artifact $\xrightarrow{d}$ artifact	a T-algebra

Notice that in the picture 'artifact' has replaced 'universe', to stress that one is now dealing with 'culture' (as opposed to 'nature') and, which is far more relevant, that the arrow has been *reversed*. Formally, an *assembly process* is a T-algebra. As a function this amounts to a collection of *constructors*. Because an artifact cannot be built simultaneously in two different ways (*i.e.*, constructors), the external structure or shape of the toolbox is usually a coproduct and the algebra arises as an *either* of constructors. For example, for the binary tree 'artifact', the suitable toolbox will be

$$\text{🧰}\; D \;=\; \mathbf{1} + Data \times D \times D$$

This means that binary trees can either be built via a constant constructor, yielding the trivial, empty tree, or via the aggregation of some data to two previously constructed trees, thus building a larger one. An assembly process arises then as

$$[empty, node] : \square\square\ D \longrightarrow D$$

Of course, 'assembly processes' can be related and compared. In fact, the notion of an algebra for a functor generalises the classical concept of an algebra; morphisms of the former also generalise classical homomorphisms as functions preserving construction. Again, if the toolbox is smooth enough, there exist a canonical representative of the assembly process: the *initial* algebra, which may be regarded as the formal analog to the 'smallest' machine able to produce all possible $\square\square$-artifacts.

In the next two sections algebraic and coalgebraic structures will be reviewed in deeper detail. As expected, *initial* algebras turn out to be *inductive data types*, *i.e.*, abstract descriptions of data structures. Dually, *final* coalgebras entail a notion of *behaviour types*, representing the dynamics of systems. Both of these structures, referred to as *categorical data types*, may be directly used in programming.

2. Algebraic Structures

9. Functors (§2.14) provide a sensible abstraction for the somewhat more vague notion of a *type* of a mathematical structure. And, in particular, for the even vaguer concept of a module *interface* in programming. Following the intuitions in the previous section, they specify the kind of 'lenses' available and the contents of 'toolboxes'. There are two main reasons for this. First, the type of a structure depends normally on types of other (sub-)structures and the very definition of a functor conveys this idea of *parameterization*. Secondly, a functor characterises, not only the structure itself, but also the transformations which preserve it. In fact, the action of a functor on a morphism f applies the transformation embodied in f to all the elements 'built into' the structure without changing the shape of the structure itself.

In this section, we will review structures intended to 'store' data elements in particular configurations. It turns out that the shapes (or types) of such configurations are suitably described by functors and the structures themselves arise as *algebras* for such functors.

10. POLYNOMIAL FUNCTORS. Despite of the wide scope of the previous paragraph, we shall restrict our attention in this thesis to a particularly well-behaved class of endofunctors. We start by considering the so-called *polynomial functors*. The class is inductively defined as the least collection of functors containing the identity Id and constant functors $\underline{K}$ for all objects K in the category, closed by functor composition and finite application of product (§2.26) and coproduct (§2.29) functors. The terminology arises from the fact that, in a distributive category, any polynomial functor can be written in the form

$$\mathsf{T}\, X \;=\; \sum_i C_i \times X^i$$

for i a natural number and C_i a constant coefficient.

Polynomial functors are standard in presenting algebraic signatures. In this thesis, however, we will need to extend the inductive definition above to include

$$\begin{aligned} \mathsf{T}\, X &= X^A && \text{(the covariant exponential functor (§2.33))} \\ \mathsf{T}\, X &= \mathcal{P}X && \text{(the finite powerset functor)} \end{aligned}$$

Functors in this class will be referred to in the sequel as *extended polynomial*. A more general class of functors — called *regular* — is obtained by further extending the definition to include *type* functors (§49).

11. ALGEBRAS. Syntactically, a data structure is described by a set of operations which specify how its values are to be produced. A *sequence*, for example, is either empty or built by adding an element to the front of a pre-existing sequence. A binary tree signature includes an empty constant and a node constructor whereby data and two other trees are aggregated to become the root node of a new tree, and so on. Notice that these two examples can be modeled by polynomial (§10) functors, which are basically n-ary sums (of alternatives) of m-ary products (of information associated to each alternative). For example,

$$\begin{aligned} \mathsf{T}_{\mathsf{Nat}}\, X &= \mathbf{1} + X && \textit{natural numbers} \\ \mathsf{T}_{\mathsf{Seq}}\, X &= \mathbf{1} + \mathit{Data} \times X && \textit{sequences} \\ \mathsf{T}_{\mathsf{Bin}}\, X &= \mathbf{1} + \mathit{Data} \times X \times X && \textit{binary trees} \\ \mathsf{T}_{\mathsf{Lef}}\, X &= \mathit{Data} + X \times X && \textit{leaf trees} \end{aligned}$$

All constructors of a given type can be grouped together into a single operation. For example, the constructors of a sequence are

$$[\mathsf{nil}, \mathsf{cons}] : \mathbf{1} + \mathit{Data} \times X \longrightarrow X$$

In general, if the shape of one of these structures is specified by a functor T, the structure itself is given as a map

$$d : \mathsf{T}\, D \longrightarrow D$$

i.e., as a T-*algebra*. Concrete structures are, therefore, obtained by specifying both the carrier set D and map d. Formally, we define,

12. Definition. For a given endofunctor T, a T-*algebra* is a pair $\langle D, d\rangle$ consisting of an object D, referred to as the *carrier* of d, and a map $d : \mathsf{T}\ D \longrightarrow D$. A T-algebra *morphism*, or simply, a T-*morphism*, between two T-algebras d and e is a map h between their carriers such that the following diagram commutes,

$$\begin{tikzcd}
\mathsf{T}\ D \arrow[r, "d"] \arrow[d, "\mathsf{T}\ h"'] & D \arrow[d, "h"] \\
\mathsf{T}\ E \arrow[r, "e"] & E
\end{tikzcd}$$

T-algebras and T-morphisms form a category C^{T} where both composition and identities are inherited from C. In the sequel, unless explicitly mentioned, we will be working on $\mathsf{Set}^{\mathsf{T}}$.

13. Compatible Relations. A basic relation that can be established between two T-algebras is one that preserves their shape. Formally, given two T-algebras $\langle D, d\rangle$ and $\langle E, e\rangle$, a *compatible relation* R is a relation on $D \times E$ that can itself be extended to a T-algebra ρ such that the canonical projections become T-algebra morphisms. This may be expressed by the commutativity of the following diagram:

$$\begin{tikzcd}
\mathsf{T}\ D \arrow[d, "d"'] & \mathsf{T}\ R \arrow[l, "\mathsf{T}\ \pi_1"'] \arrow[r, "\mathsf{T}\ \pi_2"] \arrow[d, "\rho"'] & \mathsf{T}\ E \arrow[d, "e"] \\
D & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & E
\end{tikzcd}$$

It is well known that both the *kernel* and the *graph* of a T-algebra morphism are compatible relations. Conversely, if the graph of a map h between the carriers of two algebras is a compatible relation, then h is a T-algebra morphism. Should such a relation be also an equivalence, it would be called a (T-)*congruence*, a concept which plays a fundamental role in algebraic specification.

14. Initial Algebras. As we have mentioned in §8, there is a particular T-algebra which is canonical in the sense that only equally constructed elements can be identified. This is, of course, the *term* algebra, which happens to be the initial object (§2.6) in C^{T}. Notice, by the way, that the final object in this category always exists (if C itself has a final object $\mathbf{1}$): the trivial T-algebra $\langle \mathbf{1}, ! : \mathsf{T}\ \mathbf{1} \longrightarrow \mathbf{1}\rangle$. The initial algebra will be denoted by

$$\alpha_{\mathsf{T}} : \mathsf{T}\ \mu_{\mathsf{T}} \longrightarrow \mu_{\mathsf{T}}$$

Being initial means that there exists a *unique* T-algebra morphism from α_{T} to each other T-algebra $\langle D, d\rangle$. This morphism depends, of course, on d and is said to be the *inductive extension* of d [TR98] or the *catamorphism* generated by d [MFP91]. It is written as $(\![d]\!)_{\mathsf{T}}$ or, simply, $(\![d]\!)$, if the functor is clear from the context. As any other universal construction, it is unique up to isomorphism, which justifies the particle *the* used above.

Formally, a catamorphism is characterised as the unique T-morphism making the following diagram to commute

$$\begin{tikzcd}
\mathsf{T}\ D \arrow[r, "d"] & D \\
\mathsf{T}\ \mu_{\mathsf{T}} \arrow[u, "\mathsf{T}\ (\![d]\!)_{\mathsf{T}}"] \arrow[r, "\alpha_{\mathsf{T}}"] & \mu_{\mathsf{T}} \arrow[u, "(\![u]\!)_{\mathsf{T}}"']
\end{tikzcd}$$

or, alternatively, by the following universal property:

$$k = (\![d]\!)_{\mathsf{T}} \quad \Leftrightarrow \quad k \cdot \alpha_{\mathsf{T}} = d \cdot \mathsf{T}\ k \tag{3.1}$$

15. CATA LAWS. From the universal property of catamorphisms (3.1) the following results are easily derived:

$$(\![d]\!) \cdot \alpha_{\mathsf{T}} \;=\; d \cdot \mathsf{T}\ (\![d]\!) \tag{3.2}$$

$$(\![\alpha_{\mathsf{T}}]\!) \;=\; \mathsf{id}_{\mu_{\mathsf{T}}} \tag{3.3}$$

$$h \cdot (\![d]\!) \;=\; (\![e]\!) \quad \text{if} \quad h \cdot d \;=\; e \cdot \mathsf{T}\ h \tag{3.4}$$

Equations (3.2), (3.3) and (3.4) above are usually referred to as, respectively, the *cancellation*, *reflection* and *fusion* laws for catamorphisms.

16. INDUCTION. Like any other universal construction, a catamorphism entails an *existence* property and an *uniqueness* one. Existence gives us a *definition* principle: to define a (circular) function from an initial algebra amounts to equip the target set with an algebra structure as well. In other words, a function is defined by providing a specification of its output for each of the constructors.

Uniqueness, on the other hand, gives a *proof* principle. Suppose we want to prove a predicate P over an initial algebra, *i.e.*, that the set defined by P coincides with μ_{T}. If it is possible to prove that the inclusion $i : P \hookrightarrow \mu_{\mathsf{T}}$ is a T-algebra morphism, we are done. In fact, we have equipped P with a T-algebra structure and, therefore, the composite of the corresponding catamorphism with i is unique and, necessarily, coincides with $\mathsf{id}_{\mu_{\mathsf{T}}}$. More simply, this amounts to prove that P is *closed* under the algebra constructors, which is easily recognised as the familiar *structural induction* principle.

Another way of stating this is to note that initial algebras have no proper subalgebras, which again is a direct consequence of initiality. Some intuition on the (quite special) nature of initial algebras is gathered from the following result:

17. LEMMA. The equality relation $\Delta_{\mu_{\mathsf{T}}}$ is the least T-compatible relation definable on the initial algebra α_{T}. That is, 'equality equals compatibility'.

Proof. Let $\langle R \subseteq \mu_{\mathsf{T}} \times \mu_{\mathsf{T}}, \rho\rangle$ be a compatible relation. Its projections are T-morphisms, but, by initiality,

$$\pi_1 \cdot (\!|\rho|\!)_{\mathsf{T}} \;=\; \pi_2 \cdot (\!|\rho|\!)_{\mathsf{T}} \;=\; \mathsf{id}_{\mu_{\mathsf{T}}}$$

Therefore, $\Delta_{\mu_{\mathsf{T}}} \subseteq R$, for any R. The result follows from the (trivial) fact that $\Delta_{\mu_{\mathsf{T}}}$ is itself a T-compatible relation.

□

18. 'ALGEBRAS' ARE ALGEBRAS. An expected, but fundamental, observation is that the notion of a T-algebra over Set subsumes the its classical homonym in Universal Algebra. Classically, an algebra is defined as a set plus a (finite) collection of constructors $op_i : X \times X \times ... \times X \longrightarrow X$. Denoting by ar_i the arity of operator op_i, this can be described as an algebra for functor

$$\mathsf{T}\,X \;=\; \sum_{i \in I} X^{\mathsf{ar}_i}$$

which captures its *signature*. Clearly, a T-morphism is just a classical homomorphism for the given class of algebras. Then the free term algebra over a set V of variables arises simply as the initial algebra, not for T, but for $\mathsf{T}' = \mathsf{T} + V$. T'-algebras have the form $[d, f] : \mathsf{T}\; D + V \longrightarrow D$, for any carrier D. Let

$$\langle W_V, \alpha_{\mathsf{T}'}\rangle \quad \text{with} \quad \alpha_{\mathsf{T}'} \;=\; [\sigma, i_V]$$

be the initial T'-algebra, where W_V is, as usual, the set of terms with variables taken from V, i_V the inclusion of variables from V as terms, and $\sigma : \mathsf{T}\; W_V \longrightarrow W_V$ the term formation operation, obviously a T-algebra. Given another T-algebra $d : \mathsf{T}\; D \longrightarrow D$ and a valuation function $f : V \longrightarrow D$ on the carrier of d, the induced unique morphism from the free algebra arises as catamorphism $h = (\!|[d, f]|\!)_{\mathsf{T}'}$. Therefore,

$$\begin{array}{ll}
 & h \;=\; \llparenthesis [d,f] \rrparenthesis_{\mathsf{T}'} \\
\equiv & \qquad \{ \text{ law (3.1) } \} \\
 & h \cdot \alpha_{\mathsf{T}'} \;=\; [d,f] \cdot \mathsf{T}' \; h \\
\equiv & \qquad \{ \text{ definitions } \} \\
 & h \cdot [\sigma, i_V] \;=\; [d,f] \cdot (\mathsf{T} \; h + \mathsf{id}) \\
\equiv & \qquad \{ \; + \text{ fusion and absorption } \} \\
 & [h \cdot \sigma, h \cdot i_V] \;=\; [d \cdot \mathsf{T} \; h, f]
\end{array}$$

which (omitting variable injection operators and the superscripts in $op_i^{\alpha_{\mathsf{T}'}}$, the interpretation of constructor op_i in $\alpha_{\mathsf{T}'}$) can be re-written in the more familiar format:

$$\begin{array}{rcl}
h \; op_i \langle t_1, ..., t_n \rangle & = & op_i^d \langle h \; t_1, ..., h \; t_n \rangle \\
h \; v & = & f \; v
\end{array}$$

where op_i^d stands for the interpretation in d of constructor op_i. It follows that, for a fixed d, every valuation function f corresponds bijectively to a T-morphism from $\langle W_V, \sigma \rangle$ to $\langle D, d \rangle$. And this for every algebra d. Moreover, this bijection is *natural* both in V and $\langle D, d \rangle$. In other words, and recalling §2.30, the forgetful functor $\mathsf{U} : \mathsf{C}^{\mathsf{T}} \longrightarrow \mathsf{C}$, which sends an algebra to its carrier, has a left adjoint $\mathsf{Free}^{\mathsf{T}} : \mathsf{C} \longrightarrow \mathsf{C}^{\mathsf{T}}$ mapping each set V into $\langle W_V, \sigma \rangle$. That is,

$$\mathsf{Free}^{\mathsf{T}} \;\vdash\; \mathsf{U}$$

As left adjoints preserve colimits (§2.32), it turns out that, if C has an initial object and $\mathsf{Free}^{\mathsf{T}}$ exists, $\mathsf{Free}^{\mathsf{T}} \; \emptyset$ is the initial object in C^{T}, *i.e.*, the initial algebra. In Set, its carrier is, of course, $W_\emptyset$, the set of closed terms.

3. Coalgebraic Structures

19. FROM ALGEBRAS TO COALGEBRAS. Once known how to build a data structure, one can reverse its 'assembly process'. Taking, as in §11, the simple case of sequences, such a decomposition is performed by the familiar *selectors* head and tail, for nonempty sequences, which can be joined together in

$$\langle \mathsf{head}, \mathsf{tail} \rangle : X \longrightarrow Data \times X$$

This reversal of our point of view (recall the introductory discussion in section 1) yields a different understanding of what X may stand for. First notice that what is structured in $\langle \mathsf{head}, \mathsf{tail} \rangle$ is its target, instead of its source as before. Target product

$Data \times X$ captures the fact that both the head and the tail of a sequence may be selected (or *accessed*, or *observed*) simultaneously. The emphasis on observation opens a broader understanding of the structure being defined. In fact, once one is no longer focused on how to construct X, but simply on what can be observed from it, finiteness is no longer required. Therefore, X can be more accurately thought of as a state space of a machine generating an infinite sequence of values of type $Data$. Elements of X can no longer be distinguished by construction, but should rather be identified when generating the same infinite sequence. That is, when it becomes impossible to distinguish them by the observations allowed by the shape (or 'lens') T.

Infinite sequences are common in programming. In practice they are represented by a particular *state* in a particular *state machine*. Formally, by an element of the carrier of a particular *coalgebra*, as described next.

20. DEFINITION. Given an endofunctor T, a T-*coalgebra* is a pair $\langle U, p\rangle$ consisting of an object U, referred to as the *carrier* of p, and a map $p : U \longrightarrow \mathsf{T}\ U$. A T-coalgebra *morphism*, or simply, a T-*comorphism* between two T-coalgebras, $\langle U, p\rangle$ and $\langle V, q\rangle$, is a map h between carriers U and V such that the following diagram commutes:

$$\begin{tikzcd}
U \arrow[r, "p"] \arrow[d, "h"'] & \mathsf{T}\ U \arrow[d, "\mathsf{T}\ h"] \\
V \arrow[r, "q"] & \mathsf{T}\ V
\end{tikzcd}$$

T-coalgebras and T-comorphisms form a category C_{T} where both composition and identities are inherited from C.

21. UNIVERSAL COALGEBRA. The study of coalgebras along the lines of Universal Algebra, initiated by J. Rutten in [Rut95] and [Rut96] (of which a revised version [Rut00] appeared recently), assumes coalgebra carriers' are sets, and, therefore, constitutes an exploration of $\mathsf{Set}_{\mathsf{T}}$, for different Set endofunctors T. It should be mentioned, however, that both the study of concrete coalgebras over different base categories [TR98, Mon00] and the development of Set-independent, *i.e.*, purely categorical, presentations of coalgebra theory (see, among others [TR98, PW98, GS98] and chapter 1 in A. Kurz thesis [Kur01]) have emerged as active research areas.

22. BISIMULATION. The role of bisimulations in coalgebra theory is similar to that of compatible relations (§13) in algebras. Informally, two states of a T-coalgebra (or of two different T-coalgebras) are related by a bisimulation if their observation produces equal results and this is maintained along all possible transitions. I. e., each one can mimic the other's evolution. The notion was introduced in process calculi by [Par81] and [Mil80] to capture a particular kind of observational equivalence. Later [AM88] gave a categorical definition of bisimulation which applies, not only to the kind of transition systems underlying the operational semantics of process calculi, but also to arbitrary coalgebras. As anticipated in §7, bisimulation *acquired a shape*. Formally,

23. DEFINITION. Given two T-coalgebras $\langle U, p\rangle$ and $\langle V, q\rangle$, for a Set endofunctor T, a T-*bisimulation* is a relation $R \subseteq U \times V$ which can be extended to a coalgebra ρ such that projections π_1 and π_2 lift to T-comorphisms, as expressed by the commutativity of the following diagram:

$$\begin{tikzcd}
U \arrow[d, "p"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\rho"'] & V \arrow[d, "q"] \\
\mathsf{T}\,U & \mathsf{T}\,R \arrow[l, "\mathsf{T}\,\pi_1"'] \arrow[r, "\mathsf{T}\,\pi_2"] & \mathsf{T}\,V
\end{tikzcd}$$

The definition generalises to an arbitrary base category C, replacing relation R by a monic span $\langle R, r_1, r_2\rangle$ in C whose legs lift to T-coalgebra morphisms, or, in other words, such that there is a T-coalgebra structure $\rho : R \longrightarrow \mathsf{T}R$ making the diagram below to commute:

$$\begin{tikzcd}
& R \arrow[dl, "r_1"'] \arrow[dr, "r_2"] \arrow[dd, "\rho"'] & \\
U \arrow[dd, "p"'] & & V \arrow[dd, "q"] \\
& \mathsf{T}R \arrow[dl, "\mathsf{T}r_1"'] \arrow[dr, "\mathsf{T}r_2"] & \\
\mathsf{T}U & & \mathsf{T}V
\end{tikzcd}$$

Notice that a span $\langle R, r_1, r_2\rangle$ is *monic* iff, for any arrows $f, g : X \longrightarrow R, r_1 \cdot f = r_1 \cdot g$ and $r_2 \cdot f = r_2 \cdot g$ implies $f = g$. Should C have binary products, being a monic span equivales to require split $\langle r_1, r_2\rangle : X \longrightarrow U \times V$ to be monic as well.

24. BISIMULATIONS AND COMORPHISMS. Bisimulations provide a 'relational' view of comorphisms. In fact, the graph — $\mathsf{graph}\, h$ — of a T-comorphism $h : p \longrightarrow q$ is a T-bisimulation [Rut00]. An immediate, but fundamental, corollary of this result is the fact that, in every coalgebra $\langle U, p\rangle$, the diagonal Δ_U is a bisimulation. This follows from $\Delta_U = \mathsf{graph}\ \mathsf{id}_U$, the identity id_U being trivially a comorphism.

25. PROPERTIES. [Rut00] proves several results on bisimulations in Set_T. In particular, it is shown that the *converse* of a bisimulation and, if T preserves weak pullbacks, the *composition* of two bisimulations are still bisimulations. As a corollary, the (relational) *direct* and *inverse* images of a bisimulation, as well as the *kernel* of a comorphism, are bisimulations as well. To see this, it is enough to look at their definitions, where $\circ$ denotes relational composition (§2.4),

$$\begin{aligned} h\,[R] &= (\mathsf{graph}\,h)^\circ \circ R \circ \mathsf{graph}\,h \\ h^\circ\,[R] &= \mathsf{graph}\;h \circ R \circ (\mathsf{graph}\,h)^\circ \\ \mathsf{ker}\,h &= \mathsf{graph}\;h \circ (\mathsf{graph}\,h)^\circ \end{aligned}$$

and conclude by the property above and §24. Moreover, as $\mathsf{ker}\,h$ is transitive, it gives rise to an equivalence bisimulation.

26. REMARK. Some constructions in C_T depend on extra properties of functor T. As just mentioned, *preservation of weak pullbacks* is one of them. Recall that a *weak* universal is a construction which shares all the properties of the standard one but uniqueness. Its role is crucial namely in the proofs that composition of bisimulations and kernels of comorphisms are still bisimulations as well as to state the existence of a greatest bisimulation (§28). Also notice its use in the proof of the 'full abstraction lemma' in §35.

Another important property, in Set_T, is *boundedness*. This essentially means that there exists a set B such that, for any coalgebra $\langle U, p\rangle$, the size of any sub-coalgebra (§30) of p generated by any single element $u \in U$, is bounded by the size of B. This is a rather technical condition to avoid cardinality problems, namely, when discussing existence conditions for final coalgebras (§33). Both conditions hold for the extended polynomial functors defined in §10 and used in this thesis [Rut00].

27. A CATEGORY OF BISIMULATIONS. Bisimulations on coalgebras $\langle U, p\rangle$ and $\langle V, q\rangle$ form a category $\mathsf{Bs}(p, q)$ whose arrows $m : \langle R, r_1, r_2\rangle \longrightarrow \langle S, s_1, s_2\rangle$ are morphisms $m : R \longrightarrow S$, in the base category C, such that $r_1 = s_1 \cdot m$ and $r_2 = s_2 \cdot m$, *i.e.*, the following diagram commutes:

$$\begin{tikzcd} & R \arrow[ld, "r_1"'] \arrow[rd, "r_2"] \arrow[dd, "m"'] & \\ U & & V \\ & S \arrow[lu, "s_1"] \arrow[ru, "s_2"'] & \end{tikzcd}$$

28. T-BISIMILARITY. Clearly, every morphism m in $\mathsf{Bs}(p, q)$ is a monic in C.

Proof. Let m as in §27 and f, g stand for two arbitrary C morphisms with codomain R. Then,

$$\begin{array}{ll} & m \cdot f = m \cdot g \\ = & \qquad \{ \text{ Leibniz } \} \\ & s_1 \cdot m \cdot f = s_1 \cdot m \cdot g \;\text{ and }\; s_2 \cdot m \cdot f = s_2 \cdot m \cdot g \\ = & \qquad \{ \; m \text{ is a } \mathsf{Bs}(p, q) \text{ morphism } \} \\ & r_1 \cdot f = r_1 \cdot g \;\text{ and }\; r_2 \cdot f = r_2 \cdot g \\ \Rightarrow & \qquad \{ \; \langle R, r_1, r_2 \rangle \text{ is a monic span } \} \\ & f = g \end{array}$$

□

That bisimulations are ordered by C-monics, as proved above, implies the existence of at most one arrow between any two bisimulations and generalises the well known fact that Set bisimulations are partially ordered by inclusion.

In Set the *union* of two bisimulations is also a bisimulation. Furthermore the set of all bisimulations between two coalgebras forms a complete lattice. The *greatest bisimulation* on, say, coalgebras p and q, is an *equivalence* relation, written as

$$\sim_{\langle p,q \rangle}$$

or simply, $\sim$ if the context is clear. Whenever the dependence on functor T is to be stressed, the notation $\sim^{\mathsf{T}}_{\langle p,q \rangle}$ (or $\sim^{\mathsf{T}}$) will be adopted.

29. REMARK In general, coalgebra ρ in the definition of bisimulation (§23) is not uniquely determined — a counter example for the finite powerset functor is given in [Rut00]. Uniqueness is achieved, however, for polynomial endofunctors in Set and, more generally, for functors preserving pullbacks (which, therefore, preserve monic spans). Such a lack of uniqueness makes difficult the definition of constructions like the union of bisimulations or the greatest bisimulation: some conditions on either C or T are needed to show the existence of not uniquely defined structures. For example, to obtain greatest bisimulations one may recur to either the above mentioned preservation of weak pullbacks by T or the fact that all epis are split in C (see [Rut00, Kur01]).

This explains why, in more generic approaches to coalgebra theory, some alternatives to the notion of a bisimulation have been proposed. A beautiful one consists of replacing, in the definition of bisimulation, 'monic spans' by 'epi cospans', a cospan being simply a pair of arrows with a common codomain. This leads to the definition of what is called a *cocongruence* in [Kur01] or a *compatible correlation* in [Wol00] between two coalgebras $\langle U, p\rangle$ and $\langle V, q\rangle$. It is given by an epi copsan $\langle R, r_1, r_2\rangle$ in C whose legs lift to comorphisms as expressed by the commutativity of

$$\begin{tikzcd}
 & R \arrow[dd, "\rho"] & \\
U \arrow[ur, "r_1"] \arrow[dd, "p"'] & & V \arrow[ul, "r_2"'] \arrow[dd, "q"] \\
 & \mathsf{T}R & \\
\mathsf{T}U \arrow[ur, "\mathsf{T}r_1"] & & \mathsf{T}V \arrow[ul, "\mathsf{T}r_2"']
\end{tikzcd}$$

The interest of this definition is that ρ above is uniquely determined and, moreover, the greatest cocongruence on two coalgebras exists under rather general conditions [Kur01]. Intuitively, cocongruences capture behavioural equivalence because, given two states $u \in U$ and $v \in V$, $r_1\ u = r_2\ v$ only if the behaviour they unfold to is the same, as comorphisms preserve behaviour (§24).

Such a relation E on $U \times V$ determined by a particular cocongruence (*i.e.*, $\langle u, v\rangle \in E$ iff $r_1\ u = r_2\ v$) is a bisimulation if T preserves weak pullbacks, but seems more appropriate to capture behavioural equivalence on more general situations — see [AM88] and, mainly, [Kur01] for a full discussion.

As we shall restrict ourselves in this thesis to a particularly well behaved class of Set endofunctors (§10), bisimulations provide all we need and this paragraph can be taken just as a curiosity. We would like to stress, however, the 'beauty' of the definition above which is the true dual of that of compatible relations on algebras mentioned in §13 (compare the diagrams!). As [Kur01] notes, the two crucial tools in algebra and coalgebra — compatible relations and correlations — can be simply and dually defined as, respectively, monic spans in C^{T} and epi cospans in C_{T}.

30. SUB-COALGEBRA. Let $\langle U, p\rangle$ be a T-coalgebra. A subset $i : U' \hookrightarrow U$ generates a *sub-coalgebra* of p, if the inclusion i lifts to a T-comorphism. Note that the coalgebraic structure associated to i is uniquely determined.

Proof. Consider the following diagram:

$$\begin{tikzcd}
U \arrow[r, "p"] & \mathsf{T}\,U \\
U' \arrow[u, hook, "i"] \arrow[r, "q"] & \mathsf{T}\,U' \arrow[u, hook, "\mathsf{T}\,i"']
\end{tikzcd}$$

Note that $\mathsf{T}\ i$ is either the empty map, if $U' = \emptyset$, and we are done as q arises as the initial coalgebra, or a mono in all other cases, because functors preserve split monos (§2.11) with non empty domain (incidentally, they also preserve split epis). Now notice that q, as a Set-arrow, factorizes $p \cdot i$ through a mono (*i.e.*, $\mathsf{T}\ i$). The result follows from the fact that, in such conditions, the factorisation is unique.

□

Sub-coalgebras are related to bisimulations through the following result: a subset U' of U generates a sub-coalgebra iff $\Delta_{U'}$ is a bisimulation. Note also the following characterisation of monos and epis in Set:

31. EPIS, MONOS, ISOS. Both epi and monomorphisms in Set lift to epi and monomorphisms in $\mathsf{Set}_{\mathsf{T}}$. Consequently, isomorphisms lift as well. The converse holds for epis, but for monomorphisms in $\mathsf{Set}_{\mathsf{T}}$ to be also monomorphisms as Set-arrows, it is required that T preserves weak pullbacks (see [Rut00] for a proof).

32. NATURAL TRANSFORMATIONS. Given endofunctors T and R, any natural transformation (§2.17) $\sigma : \mathsf{T} \Longrightarrow \mathsf{R}$ provides a way to 'translate' T to R-coalgebras [Rut00]. In fact, σ induces a functor from C_{T} to C_{R} which maps a T-coalgebra $\langle U, p \rangle$ into a R-coalgebra $\langle U, \sigma_U \cdot p \rangle$, and is the identity on morphisms. Moreover, this functor preserves bisimulation, *i.e.*,

$$u \sim_p^{\mathsf{T}} u' \equiv u \sim_{\sigma_U \cdot p}^{\mathsf{R}} u' \tag{3.5}$$

33. FINAL COALGEBRAS. Successive observations of (or experiments with) a T-coalgebra p reveals its behavioural patterns. These are defined in terms of the results of the observers as recorded in the shape T. Then, just as the initial algebra is canonically defined over the terms generated by successive application of constructors, it is also possible to define a canonical coalgebra in terms of such 'pure' observations. Such a coalgebra is the final object (§2.6) in C_{T}, if it exists, and will be denoted by ω_{T}.

Being *final* means that there exists a *unique* T-comorphism to ω_{T} from each other coalgebra $\langle U, p \rangle$. This is called the *coinductive extension* of p [TR98] or the *anamorphism* generated by p [MFP91], and written as $[\!(p)\!]_{\mathsf{T}}$ or, simply, $[\!(p)\!]$, if the functor is clear from context. In other words, an anamorphism is defined as the unique comorphism making the following diagram to commute:

$$\begin{tikzcd}
\nu_{\mathsf{T}} \arrow[r, "\omega_{\mathsf{T}}"] & \mathsf{T}\ \nu_{\mathsf{T}} \\
U \arrow[u, "{[\!(p)\!]_{\mathsf{T}}}"] \arrow[r, "p"] & \mathsf{T}\ U \arrow[u, "{\mathsf{T}\ [\!(p)\!]_{\mathsf{T}}}"']
\end{tikzcd}$$

or, alternatively, by the following universal law:

$$k = [\!(p)\!]_{\mathsf{T}} \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p \tag{3.6}$$

For each $u \in U$, $[\!(p)\!]_{\mathsf{T}}\ u$ can be thought of as the (observable) behaviour of a sequence of p transitions starting at state u. This explains yet another alternative designation for an anamorphism: *unfold*. On its turn, u in $[\!(p)\!]_{\mathsf{T}}\ u$, is called the *seed* of the anamorphism.

As in the algebraic case, the *existence* part of the universal property provides a *definition* principle for (circular) functions to the final coalgebra which amounts to equip their source with a coalgebraic structure specifying the 'one-step' dynamics. Then the corresponding anamorphism gives the rest. In other words, such functions are defined by specifying their output under all different observers. The *uniqueness* part, on the other hand, offers a powerful *proof* principle — *coinduction* — discussed in §36.

34. ANA LAWS. The following laws follow from the universal property of anamorphisms — equation (3.6). Comparing with §15, one easily recognises them as the *cancellation*, *reflection* and *fusion* result for anamorphisms, respectively.

$$\omega_{\mathsf{T}} \cdot [\!(p)\!] \;=\; \mathsf{T}\ [\!(p)\!] \cdot p \tag{3.7}$$

$$[\!(\omega_{\mathsf{T}})\!] \;=\; \mathsf{id}_{\nu_{\mathsf{T}}} \tag{3.8}$$

$$[\!(p)\!] \cdot h \;=\; [\!(q)\!] \quad \text{if} \quad p \cdot h \;=\; \mathsf{T}\ h \cdot q \tag{3.9}$$

35. LEMMA. We are ready to state and prove the fundamental characterisation result on final coalgebras, referred to, in [TR98], as the *full abstraction theorem* for final semantics. Let T preserve weak pullbacks. Given two T-coalgebras $\langle U, p\rangle$ and $\langle V, q\rangle$, any two states $u \in U$ and $v \in V$ satisfy

$$u \sim^{\mathsf{T}}_{\langle p,q\rangle} v \quad \equiv \quad [\!(p)\!]_{\mathsf{T}}\ u = [\!(q)\!]_{\mathsf{T}}\ v \tag{3.10}$$

Proof.

$\Rightarrow$
Let $\langle R \subseteq U \times V, \rho\rangle$ be a bisimulation such that $\langle u, v\rangle \in R$. Then, projections π_1 and π_2 lift to T-comorphisms, *i.e.*, $\pi_1 : \langle R, \rho\rangle \longrightarrow \langle U, p\rangle$ and $\pi_2 : \langle R, \rho\rangle \longrightarrow \langle V, q\rangle$. Now, composites $\llbracket p \rrbracket_{\mathsf{T}} \cdot \pi_1$ and $\llbracket q \rrbracket_{\mathsf{T}} \cdot \pi_2$ are comorphisms to the final coalgebra with identical source. By finality, they coincide.
$\Leftarrow$
We begin by defining relation $R = \{\langle u, v\rangle \in U \times V | \ \llbracket p \rrbracket_{\mathsf{T}}\ u = \llbracket q \rrbracket_{\mathsf{T}}\ v\}$. All we have to do is to prove that R lifts to a bisimulation, *i.e.*, we have to equip R with a coalgebraic structure ρ such that projections lift to comorphisms. Relation R is, in fact, the pullback (in Set) of $\llbracket p \rrbracket_{\mathsf{T}}$ and $\llbracket q \rrbracket_{\mathsf{T}}$. As T preserves weak pullbacks (§31), the following diagram is also, at least, a weak pullback:

$$\begin{tikzcd}
\mathsf{T}\, R \arrow[r, "\mathsf{T}\ \pi_1"] \arrow[d, "\mathsf{T}\ \pi_2"'] & \mathsf{T}\, U \arrow[d, "\mathsf{T}\ \llbracket p \rrbracket_{\mathsf{T}}"] \\
\mathsf{T}\, V \arrow[r, "\mathsf{T}\ \llbracket q \rrbracket_{\mathsf{T}}"] & \mathsf{T}\ \nu_{\mathsf{T}}
\end{tikzcd}$$

Now notice that R can be made a cone over the diagram for which $\mathsf{T}\ R$ is a weak pullback:

$$\begin{tikzcd}
R \arrow[dr, dotted, "\rho"] \arrow[drr, bend left, "p \cdot \pi_1"] \arrow[ddr, bend right, "q \cdot \pi_2"'] & & \\
& \mathsf{T}\, R \arrow[r, "\mathsf{T}\ \pi_1"] \arrow[d, "\mathsf{T}\ \pi_2"] & \mathsf{T}\, U \arrow[d, "\mathsf{T}\ \llbracket p \rrbracket_{\mathsf{T}}"] \\
& \mathsf{T}\, V \arrow[r, "\mathsf{T}\ \llbracket q \rrbracket_{\mathsf{T}}"] & \mathsf{T}\ \nu_{\mathsf{T}}
\end{tikzcd}$$

Here ρ plays the role of a mediating morphism to the weak pullback. It is, of course, not necessarily unique, but this is not required in the definition of a bisimulation. Moreover, combining this with the first part of the theorem, we conclude that $\langle R, \rho\rangle$ is, not only, a bisimulation, but the greatest one, *i.e.*, it coincides with $\sim^{\mathsf{T}}_{\langle p,,q\rangle}$.

□

Finally notice that, although the proof was presented in Set, the argument extends to an arbitrary category C. It suffices to recall that, in general, a T-bisimulation is defined as a span whose legs lift to comorphisms (§23) and define R directly as the pullback of the two anamorphisms. The general result stating that if T preserves weak pullbacks, then any pullback in C lifts to a bisimulation, is due to Aczel and Mendler [AM88].

36. COINDUCTION. The previous result shows that the final coalgebra ω_{T} acts as a state *classifier* for any other T-coalgebra and, moreover, that bisimulation provides a *local* proof theory for behavioural equality. This is exactly the core of the *coinduction* principle which may be stated, for every bisimulation R on a coalgebra $\langle U, p\rangle$ satisfying it,

$$R \subseteq \Delta_{\mathsf{id}_U}$$

Therefore, in such a coalgebra, to prove the equality of two states, it is enough to find a bisimulation containing them. Coalgebras that satisfy the *coinduction* principle are called *simple*. An alternative, equivalent, characterisation stresses the fact that Δ_{id_U} is the greatest bisimulation in such a coalgebra. This is indeed the case of the final coalgebra, since the projections of any bisimulation $\langle R \subseteq \nu_{\mathsf{T}} \times \nu_{\mathsf{T}}, \rho\rangle$ are comorphisms to ω_{T} and, therefore, by finality, $\pi_1 = [\![(\rho)]\!]_{\mathsf{T}} = \pi_2$.

37. COFREE COALGEBRAS. Let us open a parenthesis to investigate the coalgebraic dual to *free* algebras discussed in §18. Free algebras are initial algebras with additional terms to represent variables. Dually, a T-*cofree* coalgebra [Jac95] is a final T-coalgebra in which states (thought of as behaviours) are *coloured* by elements of a set V (recall the informal discussion on 'lenses' in the introduction to this chapter). Formally, they are final coalgebras for functor $\mathsf{T}' = \mathsf{T} \times V$, *i.e.*,

$$\langle \sigma, \epsilon\rangle : K_V \longrightarrow \mathsf{T}\ K_V \times V$$

where $\epsilon : K_V \longrightarrow V$ is the state colouring morphism. Suppose, now, we are given a T-coalgebra $p : U \longrightarrow \mathsf{T}\ U$ and a 'colour assignment' $f : U \longrightarrow V$ from the carrier of p. The induced unique morphism to the cofree coalgebra is anamorphism $h = [\![(\langle p, f\rangle)]\!]_{\mathsf{T}'}$. This is, in fact, an extension of the 'colour assignment' to behaviours generated from p, as a simple calculation shows,

$$\begin{aligned}
& h \;=\; [\![(\langle p, f\rangle)]\!]_{\mathsf{T}'} \\
\equiv \quad & \{\ \text{ana universal (3.6)}\ \} \\
& \omega_{\mathsf{T}'} \cdot h \;=\; \mathsf{T}'\ h \cdot \langle p, f\rangle \\
\equiv \quad & \{\ \text{definitions}\ \} \\
& \langle \sigma, \epsilon\rangle \cdot h \;=\; (h \times \mathsf{id}_)\cdot \langle p, f\rangle \\
\equiv \quad & \{\ \times\text{-fusion and absorption}\ \} \\
& \langle \sigma \cdot h, \epsilon \cdot h\rangle \;=\; \langle h \cdot p, f\rangle
\end{aligned}$$

This can be re-written as a *coinductive definition* of h:

$$\begin{aligned}
\sigma\ (h\ u) \;&=\; h\ (p\ u) \\
\epsilon\ (h\ u) \;&=\; f\ u
\end{aligned}$$

This is an example of definition by coinduction: the function being defined, h, is specified by analysing its value under each observer. Considering the common case in which the most external shape of T is a product with n factors, the first clause unfolds to a collection of equations, one for each observer ob_i, with $i \in n$,

$$ob_i\ (h\ u) \ = \ h\ (ob_i{}^p\ u)$$

Similarly to what happens in the free algebra case, it turns out that there exists a bijective correspondence between arrows $f : U \longrightarrow V$ in C and T-comorphisms from $\langle U, p\rangle$ to $\langle K_V, \sigma\rangle$, the last being obviously a T-coalgebra. Being natural both in V and $\langle U, p\rangle$, this bijection witnesses an adjunction

$$\mathsf{U} \vdash \mathsf{CoFree}^{\mathsf{T}}$$

between the forgetful functor to C and the functor which associates each V to coalgebra $\langle K_V, \sigma\rangle$. As right adjoints preserve limits (§2.32), it turns out that, if C has a final object $\mathbf{1}$ and $\mathsf{CoFree}^{\mathsf{T}}$ exists, $\mathsf{CoFree}^{\mathsf{T}}\ \mathbf{1}$ is the final object in C_{T}, *i.e.*, the final coalgebra. This leads us to the question of existence of final coalgebras.

38. EXISTENCE. In most cases being aware of the existence of a final coalgebra is all one needs to know. In fact, like any other universal, the use of the final coalgebra is completely determined by the universal property rather than by the internal structure of its carrier. Note, however, that this may be contrasted with a common and fruitful procedure used in coalgebraic reasoning which consists of exhibiting the underlying final coalgebra of some mathematical objects, such as streams or languages, to apply coinduction in the study of their properties (see, for example, [Jac96a, Rut98] or [Rut01]).

The next few paragraphs discuss briefly existence of final coalgebras, to conclude that they do exist for all functors considered in this thesis. Some concrete examples of final coalgebras are mentioned in §§43 and 44. Prior to that we shall recall a well known result which characterises both final coalgebras and initial algebras as fixpoints of functors.

39. LAMBEK LEMMA. The final object in C_{T}, if it exists, is an isomorphism.

Proof. Let $\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\ \nu_{\mathsf{T}}$ be the final T-coalgebra. Because $\mathsf{T}\ \omega_{\mathsf{T}} : \mathsf{T}\ \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\ \mathsf{T}\ \nu_{\mathsf{T}}$ is a T-coalgebra as well anamorphism $[\!(\mathsf{T}\ \omega_{\mathsf{T}})\!]_{\mathsf{T}}$ exists. The composite $[\!(\mathsf{T}\ \omega_{\mathsf{T}})\!]_{\mathsf{T}} \cdot \omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \nu_{\mathsf{T}}$ is also a comorphism and, by finality, coincides with $\mathsf{id}_{\nu_{\mathsf{T}}}$. So $[\!(\mathsf{T}\ \omega_{\mathsf{T}})\!]_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = \mathsf{id}_{\nu_{\mathsf{T}}}$ and the proof is half done. For the other half, note

$$
\begin{array}{ll}
 & \omega_{\mathsf{T}} \cdot [\![(\mathsf{T}\ \omega_{\mathsf{T}})]\!]_{\mathsf{T}} \\
= & \quad \{ \text{ comorphism } \} \\
 & \mathsf{T}\ [\![(\mathsf{T}\ \omega_{\mathsf{T}})]\!]_{\mathsf{T}} \cdot \mathsf{T}\ \omega_{\mathsf{T}} \\
= & \quad \{ \text{ } \mathsf{T} \text{ functor } \} \\
 & \mathsf{T}\ ([\![(\mathsf{T}\ \omega_{\mathsf{T}})]\!]_{\mathsf{T}} \cdot \omega_{\mathsf{T}}) \\
= & \quad \{ \text{ just proved } \} \\
 & \mathsf{T}\ \ \mathsf{id}_{\nu_{\mathsf{T}}} \\
= & \quad \{ \text{ } \mathsf{T} \text{ functor } \} \\
 & \mathsf{id}_{(\mathsf{T}\ \mathsf{id}_{\nu_{\mathsf{T}}})}
\end{array}
$$

□

As isomorphisms are self-dual, this also entails that the initial algebra of a functor, if it exists, is an isomorphism. Such was the original statement of the lemma in [Bar70], where it is credited to J. Lambek. As a corollary, notice that $\omega_{\mathsf{T}}^{\circ} = [\![(\mathsf{T}\ \omega_{\mathsf{T}})]\!]_{\mathsf{T}}$.

40. FIXPOINTS. Lambek Lemma characterises both initial algebras and final coalgebras for a functor T as fixpoints of the equation

$$X \cong \mathsf{T}\, X$$

The initial algebra is said to be the *least* fixpoint of T, up to isomorphism, and the final coalgebra the *greatest*. The terminology arises from an analogy with what happens in a partial order $\langle P, \leq \rangle$ seen as a category. A functor, in such a setting, is just a monotone function and, therefore a coalgebra (respectively, an algebra) is an element x of P such that $x \leq \mathsf{T}\ x$ (respectively, $\mathsf{T}\ x \leq x$). The final coalgebra is, then, an element $m \leq \mathsf{T}\ m$ such that, for all $x \in P$, $x \leq \mathsf{T}\ x \Rightarrow x \leq m$. By Tarski's theorem, [Tar55], this is the greatest fixpoint of T with respect to $\leq$. Dually, the initial algebra arises as the least fixpoint. By analogy, least and greatest fixpoints of an endofunctor in an arbitrary category are defined as the initial algebra and the final coalgebra, respectively. As [MA86] remarks

> *This defines what we mean by least and greatest in the general case. There is no pre-established order.*

41. REMARK. The replacement of isomorphism by strict equality in an universe of sets raises foundational problems. In particular, the strict greatest fixpoint of most functors T would violate the *foundation axiom* of classical set theory which states the well-foundedness of the membership relation, *i.e.*, the no existence of infinitely descending chains ... $\in x_2 \in x_1 \in x$. This observation lead to the development of *non-well-founded* set theory [Acz88], in which the axiom is replaced by an 'anti-foundation axiom'. Such non standard set theory arose in the work of P. Aczel to provide a final semantics for Ccs processes as elements of strict final coalgebras for functor $\mathcal{P}(A \times \mathsf{Id})$ in the category Class of large sets (or classes), where $\mathcal{P}$ stands for the (unrestricted) powerset functor.

From the point of view of category theory, such a 'foundational shift' seems avoidable, to a great extent, as final coalgebras do exist in Set and Class with classical foundations, as Aczel himself proved in [Acz88, AM88]. M. Barr [Bar92], in a polemic with J. Bairwise, illustrated this point of view in the following terms:

> *It is unfortunate that such solutions* [resorting to anti-foundation] *exist, for their main effect is to avoid giving serious consideration to the real problem: the irrelevance of actual elements in mathematics.*

The theory of non-well-founded sets has, however, an interest in its own, namely on research of set theoretic foundations for corecursion; see [BM96] where several modelling applications are given, in particular in the area of artificial intelligence.

42. Construction of Final Coalgebras. There is, however, a fundamental restriction which may prevent the existence of final coalgebras: *cardinality*. In particular, Lambek lemma implies that a final coalgebra for the unrestricted powerset functor in Set cannot exist, as it would violate Cantor's theorem. The problem is avoided by moving to the large category Class, as [AM88] does, and is, of course, non existent for the finite powerset functor we have been considering. In general, cardinality restrictions are avoided if we require T to be bounded (§26). [Rut00] proves that all extended polynomial functors (§10) are indeed bounded. In this paragraph we shall briefly review the construction of final coalgebras in Set.

The general method for building final coalgebras for polynomial functors is a generalisation of Kleene's theorem for finding fixpoints in complete partial orders. Basically, a fixpoint arises as the limit of a descending chain

$$\mathbf{1} \xleftarrow{!} \mathsf{T}\,\mathbf{1} \xleftarrow{\mathsf{T}\,!} \mathsf{T}^2\,\mathbf{1} \xleftarrow{\mathsf{T}^2\,!} \cdots$$

where $\mathsf{T}^n = \mathsf{T} \cdot \mathsf{T}^{n-1}$. More concretely, this can be seen as the set

$$\{\langle x_0, x_1, x_2, \cdots \rangle \mid x_n \in \mathsf{T}^n\ \mathbf{1}\ \wedge\ (\mathsf{T}^n\,!)\ x_{n+1} = x_n \quad \text{for all } n\}$$

This method requires T to preserve limits of descending chains, a condition usually known as *ω-continuity*, which is indeed the case of polynomial functors and the covariant exponential functor. A dual requirement, and a dual procedure, computes initial algebras. Recall that the original Kleene theorem can also be used to compute both least and greatest fixpoints. See [SP82] for an early reference and [MA86] for a detailed proof and examples.

43. EXAMPLES. For the cases covered by the Kleene method we can obtain concrete descriptions of the final coalgebras. Moreover, they arise as completions of the corresponding initial algebras. Let us see some examples in Set.

- The trivial example is the final coalgebra for the identity functor Id: it is, as expected, $\langle \mathbf{1}, \mathsf{id}_{\mathbf{1}} \rangle$.
- For functor $\mathsf{T}\ X = O \times X$ the carrier of the final coalgebra is the set O^ω of infinite sequences of O, with $\langle \mathsf{head}, \mathsf{tail} \rangle$ as the dynamics. This extends to $O^\infty = O^* \cup O^\omega$, for the usual 'list' functor $\mathsf{T}\ X = \mathbf{1} + O \times X$.
- Functor $\mathsf{T}\ X = O \times X^I$, which is the type of (deterministic) systems with an observer (or attribute) o and a parametrized action a, has as final coalgebra

$$\langle \mathsf{o}, \mathsf{a} \rangle : O^{I^*} \longrightarrow O \times O^{I^* I}$$

where

$$\begin{aligned} \mathsf{o}\, m &= m\ \mathsf{nil} \\ \mathsf{a}\, m &= \lambda\, i\ \lambda\, s\ .\ m\ (s \frown \langle i \rangle) \end{aligned}$$

which amounts to infinite trees whose branches are labelled by sequences of inputs and leafs by values of O.

- The final coalgebra for the more general shape

$$\mathsf{T}\ X \;=\; \prod_j (O_j + P_j \times X)^{I_j}$$

is constructed in [Jac96b]. Its carrier is the space of functions from$\sum_j I_j$ to $(\sum_j O_j + \sum_j P_j)$, subject to an invariant that assures type compatibility (*i.e.*, an input on I_j will produce output in O_j and C_j) and completion (in the sense that when a node labelled by an output value of type O_j, for any j, is reached, the tree is completed by an infinite tree whose nodes are all labelled by that same value). Again branches are labelled by inputs and nodes by values from $\sum_j O_j + \sum_j P_j$. The root, however, is not labelled.

44. POWERSET. Kleene's theorem does not apply to the (finite) powerset functor. In this case, existence of final coalgebras has been proved by M. Barr [Bar93]. Roughly, the intuition is to take the coproduct of all T-coalgebras and, then, quotienting it by the greatest bisimulation. Because such coproduct may not exist, the argument is reformulated in terms of a set of 'generators'.

For the common case $\mathsf{T}\ X = \mathcal{P}(O \times X)$, this yields, as one would expect from the semantics of process calculi, the set of rooted finitely branching trees, with branches labelled by O, quotiented by the greatest bisimulation.

45. THE STRUCTURE OF C_T. What M. Barr actually proved in [Bar93] is that the forgetful functor $U : \mathsf{Set}_\mathcal{P} \longrightarrow \mathsf{Set}$ has a right adjoint. This is, of course, $\mathsf{CoFree}^\mathcal{P}$ (§37) and, as Set has a final object, $\mathsf{CoFree}^\mathcal{P}\ \mathbf{1}$ gives the final coalgebra. Furthermore, the paper unveils much of the structure of Set_T for an arbitrary functor T. In particular, it is shown that coproducts and coequalizers exist and their carrier coincides with the same construction in Set. [Rut00] shows a similar result for all limits that are preserved by T. The structure of C_T, in the general case, has been studied by a number of people ([Rut00, GS98, Wor98, PW98, Ada00], among others).

4. Categorical Data Types

46. INTRODUCTION. In the previous sections we have seen how algebras and coalgebras for an endofunctor T provide simple mathematical models for a variety of phenomena in programming. In particular, their *initial* and *final* representatives become suitable abstract descriptions of finite *data structures*, in the former case, and *behavioural patterns*, in the latter. They are often mentioned as *inductive* (or initial) and *coinductive* (or final) categorical types, respectively, after T. Hagino thesis [Hag87b].

Furthermore, in both cases, they come equipped with a definitional and a proof principle, arising from a core universal property. Such a property is expressed by the existence and uniqueness of a special arrow in the respective category of algebras or coalgebras. As so it can be turned into a programming *combinator* and used, not only to calculate programs, but also to program with. *Cata* and *anamorphisms* are typical examples, but, of course, not the only ones. *Paramorphisms* [Mee92] and *apomorphisms* [VU97] capture more general primitive recursion and primitive corecursion schemes; they are mentioned in appendix E. See [Fok92b] for the 'family album'.

The functional language CHARITY [CF92] provides a programming notation entirely based on such connectives. This has motivated the use of CHARITY in this thesis to prototype some of the proposed constructions on components.

The purpose of this section is twofold. First, we review how the construction of initial algebras and final coalgebras can be made functorial (§§47 and 49). Secondly, we recall a common assumption in this area: the requirement that functors should be *strong* (§50). In fact, such a requirement is basic in CHARITY. Therefore, this section details the construction of the strong versions of the above mentioned combinators as they are used in the language. This complements the more 'programming-oriented' introduction to CHARITY provided in appendix E. Note that our presentation differs from the one followed in CHARITY accompanying papers (*e.g.*, [CS92, CS95]), which resorts to a fibrational setting, perhaps less familiar to a computer science audience.

47. COTYPE FUNCTORS. Functors used to capture the shape (or signature) of the examples discussed in §43 are *parametric* on the output and input universes (represented there by O and I, respectively). The carrier of the final coalgebra, is defined in terms of observations and involves, naturally, these parameters. Even if, of course, it does not depend on which concrete sets symbols O and I stand for. This means that its construction can be made functorial, as follows.

First of all, functor T has to be expressed as a n-ary functor (§2.16), *e.g.*,

$$\begin{aligned} \mathsf{T}\ \langle O, X\rangle &= O \times X \\ \mathsf{T}\ \langle O, I, X\rangle &= O \times X^I \end{aligned}$$

from which we extract the section correspondent to the observation parameters (*e.g.*, T_O or $\mathsf{T}_{O,I}$). Without loss of generality, consider T_A, for A standing for the above mentioned parameters. This section induces both a coinductive and an inductive type, parametric on A, arising, respectively, as its final coalgebra and initial algebra. In other words (§40), as the greatest and least fixpoints to the following equation

$$X \cong \mathsf{T}\ \langle A, X\rangle$$

Following the convention established in §14 and §33, their carriers will be denoted by ν_{T_A} and μ_{T_A}, respectively. Then define a functor $\mathsf{N}_\mathsf{T} : \mathsf{C} \longrightarrow \mathsf{C}$, called the *cotype functor* for T, such that

$$\begin{aligned} \mathsf{N}_\mathsf{T}\ A &= \nu_{\mathsf{T}_A} \\ \mathsf{N}_\mathsf{T}\ f &= [\![\mathsf{T}\ (f, \mathsf{id}_{\mathsf{N}_\mathsf{T}\ A}) \cdot \omega_{\mathsf{T}_A}]\!]_\mathsf{T} \end{aligned}$$

for $f : A \longrightarrow B$. The action on morphisms of $\mathsf{N}_{\mathsf{T}_A}$ is explained by the following diagram:

$$\begin{tikzcd}
\nu_{\mathsf{T}_B} \arrow[rr, "\omega_{\mathsf{T}_B}"] & & \mathsf{T}_B\ \nu_{\mathsf{T}_B} \\
\nu_{\mathsf{T}_A} \arrow[u, "\mathsf{N}_{\mathsf{T}}\ f"] \arrow[r, "\omega_{\mathsf{T}_A}"] & \mathsf{T}_A\ \nu_{\mathsf{T}_A} \arrow[r, "\mathsf{T}\ (f, \mathsf{id}_{\mathsf{N}_{\mathsf{T}}\ A})"] & \mathsf{T}_B\ \nu_{\mathsf{T}_A} \arrow[u, "\mathsf{T}_B\ \mathsf{N}_{\mathsf{T}}\ f"']
\end{tikzcd}$$

Notice that $\mathsf{N}_{\mathsf{T}_A}\ f$ is well defined as it is the unique T-comorphism making the diagram to commute.

48. LAWS. This *cotype* functor satisfies the following laws, each of them proved by direct application of the anamorphism laws (§34):

$$\mathsf{N}_{\mathsf{T}}\ \mathsf{id}_A = \mathsf{id}_{\mathsf{N}_{\mathsf{T}}\ A} \tag{3.11}$$

$$\mathsf{N}_{\mathsf{T}}\ (g \cdot f) = \mathsf{N}_{\mathsf{T}}\ g \cdot \mathsf{N}_{\mathsf{T}}\ f \tag{3.12}$$

$$\mathsf{N}_{\mathsf{T}}\ f \cdot [\!(p)\!]_{\mathsf{T}} = [\!(\mathsf{T}\ (f, \mathsf{id} \cdot p)\!]_{\mathsf{T}} \tag{3.13}$$

The first two are the usual functoriality properties. Equation (3.13), on the other hand, is usually referred to as the *absorption* law for anamorphisms.

49. TYPE FUNCTORS. Dually, a functor assigning to each object A the initial T_A-algebra can be defined and proved to satisfy dual laws. It is called the T-*type* functor and given by:

$$\begin{aligned}
\mathsf{M}_{\mathsf{T}}\ A &= \mu_{\mathsf{T}_A} \\
\mathsf{M}_{\mathsf{T}}\ f &= (\![\alpha_{\mathsf{T}_B} \cdot \mathsf{T}\ (f, \mathsf{id}_{\mathsf{M}_{\mathsf{T}}\ B})]\!)_{\mathsf{T}}
\end{aligned}$$

for $f : A \longrightarrow B$. Again its action on morphisms arises from the following diagram:

$$\begin{tikzcd}
\mathsf{T}_A\ \mu_{\mathsf{T}_A} \arrow[rr, "\alpha_{\mathsf{T}_A}"] \arrow[d, "\mathsf{T}_A\ \mathsf{M}_{\mathsf{T}}\ f"'] & & \mu_{\mathsf{T}_A} \arrow[d, "\mathsf{M}_{\mathsf{T}}\ f"] \\
\mathsf{T}_A\ \mu_{\mathsf{T}_B} \arrow[r, "\mathsf{T}\ (f, \mathsf{id}_{\mathsf{M}_{\mathsf{T}}\ B})"] & \mathsf{T}_B\ \mu_{\mathsf{T}_B} \arrow[r, "\alpha_{\mathsf{T}_B}"] & \mu_{\mathsf{T}_B}
\end{tikzcd}$$

Type functors are extensively used in functional programming, see *e.g.*, [Bir98]. As expected they verify the duals of laws in §48.

50. STRONG TYPES. A data type is *strong* if its signature is captured by a so-called strong functor. As detailed below, a functor T is strong if it possess a (tensorial) *strength*, *i.e.*, a natural transformation

$$\tau_r^{\mathsf{T}} : \mathsf{T} \times - \Longrightarrow \mathsf{T}(\mathsf{Id} \times -)$$

subject to certain conditions. Its effect is to *distribute* context "$-$" along functor T (see §54 for a set of examples). If types are modeled in such a setting, the universal combinators (*e.g.*, cata and anamorphisms) will possess a somewhat more general shape, able to deal with the presence of extra parameters in the functions being defined. And this is possible even in case the underlying category is not cartesian closed (and therefore *currying* is not available).

There is, however, another reason supporting the strongness requirement, and probably a more fundamental one when building categorical models for, *e.g.*, functional programming. In fact, a basic requirement in such models is the need for arrows to *internalise*, for every functor capturing a type signature, its action on morphisms. Such a collection of arrows will, therefore, give semantics to a generalised *map* functional.

The task of mimicking the action of a functor T on morphisms within the (semantic) category, is again accomplished by a natural transformation σ^{T}, called the *functorial strength*. It turns out that both notions of strength are the two sides of the same coin in the sense that, for the same functor, one uniquely determines the other.

In the sequel we provide a brief introduction to both concepts and, then, quickly turn into the analysis of *strongly final* coalgebras (and *strongly initial* algebras) and their properties.

The basic results on strength can be found on [Koc72]. A standard reference on strong data types is [CS92]. A subsequent paper [CS95] describes a term logic for programming with them, which is implemented in the experimental language CHARITY [CF92].

51. FUNCTORIAL STRENGTH. As mentioned above, a *functorial strength* σ^{T} is a natural transformation which internalises the action of a functor T on morphisms. Its component, at arbitrary objects X, Y, is a map

$$\sigma^{\mathsf{T}}_{X,Y} : \mathsf{C}[X,Y] \longrightarrow \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ Y]$$

satisfying

$$\begin{tikzcd}
\mathbf{1} \arrow[r, "\mathsf{id}_{\mathsf{T}\ X}"] \arrow[d, "\mathsf{id}_X"'] & \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ X] \\
\mathsf{C}[X,Y] \arrow[ur, "\sigma^{\mathsf{T}}_{X,X}"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{C}[Y,Z] \times \mathsf{C}[X,Y] \arrow[r, "\cdot"] \arrow[d, "\sigma^{\mathsf{T}}_{Y,Z} \times \sigma^{\mathsf{T}}_{X,Y}"'] & \mathsf{C}[X,Z] \arrow[d, "\sigma^{\mathsf{T}}_{X,Z}"] \\
\mathsf{C}[\mathsf{T}\ Y, \mathsf{T}\ Z] \times \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ Y] \arrow[r, "\cdot"] & \mathsf{C}[\mathsf{T}\ X, \mathsf{T}\ Z]
\end{tikzcd}$$

Of course, if C is Cartesian closed (§2.38), homsets have internal representations as exponentials and, consequently, functorial strength assumes the more familiar form $\sigma^{\mathsf{T}}_{X,Y} : X^Y \longrightarrow \mathsf{T}\ X^{\mathsf{T}\ Y}$. The subject is far more elaborate than we can glimpse here. It arises in the context of *enriched category theory* [Kel82], a branch of research lead by the idea of *internalising*: 'meta' notions normally lying on top of a category, become (internally) represented by (families of) morphisms in the category itself. In a Cartesian closed category, a functor with functorial strength is said to be *enriched over the exponential*, which is thus another common designation for a strong functor. Notice that the axioms for functorial strength captured by the two diagrams above are no more than an internal representation of the properties of a functor: identities are preserved and the functor commutes with composition.

52. TENSORIAL STRENGTH. The notion of a tensorial strength provides an alternative way of establishing the strongness of a functor via a first-order formulation. Moreover, it explicitly offers a way of distributing 'context', and therefore, dealing with extra parameters in a computation. We review the concept below and discuss its formulation for the kind of functors used in the thesis.

Let C be a category with finite products and T an endofunctor in C. T is said to have a (*tensorial*) *strength* if there exists a natural transformation

$$\tau_r^{\mathsf{T}} : \mathsf{T} \times - \Longrightarrow \mathsf{T}(\mathsf{Id} \times -)$$

called a *right strength*, or simply a *strength*, such that, for all $I, X, Y \in \mathrm{obj}(\mathsf{C})$, equations

$$\mathsf{Tl} \cdot \tau_r = \mathsf{l} \tag{3.14}$$

$$\mathsf{Ta}^\circ \cdot \tau_r = \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \tag{3.15}$$

hold. These equations express the *unit* and the *associativity* law for τ_r and can be represented as

$$\begin{tikzcd}
\mathsf{T}I \times \mathbf{1} \arrow[r, "\tau^{\mathsf{T}}_{r_{I,\mathbf{1}}}"] \arrow[dr, "\mathsf{l}_{\mathsf{T}I}"'] & \mathsf{T}(I \times \mathbf{1}) \arrow[d, "\mathsf{T}(\mathsf{l}_{\mathsf{T}I})"] \\
 & \mathsf{T}I
\end{tikzcd}$$

$$\begin{tikzcd}
\mathsf{T}I \times (X \times Y) \arrow[rr, "\tau^{\mathsf{T}}_{r_{I,X\times Y}}"] \arrow[d, "\mathsf{a}^\circ"'] & & \mathsf{T}(I \times (X \times Y)) \arrow[d, "\mathsf{Ta}^\circ"] \\
(\mathsf{T}I \times X) \times Y \arrow[r, "\tau^{\mathsf{T}}_{r_{I,X}} \times \mathsf{id}_Y"] & \mathsf{T}(I \times X) \times Y \arrow[r, "\tau^{\mathsf{T}}_{r_{I\times X,Y}}"] & \mathsf{T}((I \times X) \times Y)
\end{tikzcd}$$

Dually, a natural transformation $\tau_l^{\mathsf{T}} : - \times \mathsf{T} \Longrightarrow \mathsf{T}(- \times \mathsf{Id})$ satisfying similar laws is called a *left strength*. Assuming T implicit, note that $\tau_l = \mathsf{Ts} \cdot \tau_r \cdot \mathsf{s}$ and

$\tau_r = \mathsf{T}\mathsf{s} \cdot \tau_l \cdot \mathsf{s}$. The unit and associativity axioms also hold for the *left strength*, in the dual form:

$$\mathsf{T}\mathsf{r} \cdot \tau_l = \mathsf{r} \tag{3.16}$$

$$\mathsf{T}\mathsf{a} \cdot \tau_l = \tau_l \cdot (\mathsf{id} \times \tau_l) \cdot \mathsf{a} \tag{3.17}$$

53. REMARK. As anticipated in §50, ${\tau_l}^{\mathsf{T}}$ (or its 'right' version) and σ^{T} can be mutually defined by

$${\tau_l}^{\mathsf{T}} = \mathsf{ev} \cdot (\sigma^{\mathsf{T}} \cdot \mathsf{sp} \times \mathsf{id}) \tag{3.18}$$

$$\sigma^{\mathsf{T}} = {\mathsf{id}^{\mathsf{T}}}^{\,\mathsf{ev} \cdot {\tau_l}^{\mathsf{T}}} \cdot \mathsf{sp} \tag{3.19}$$

where sp and ev are, respectively, the unit and counit of the product-exponential adjunction (§2.33). This correspondence is proved in [Kel82].

54. A STRENGTH CATALOGUE. First of all, there is a tensorial strength for all *polynomial* functors (§10) in any distributive category. A pointfree definition, by induction on the structure of polynomial functors, is as follows:

$$\tau^{\underline{K}}_{r_{I,X}} = \pi_1 \qquad : K \times X \to K$$

$$\tau^{\mathsf{Id}}_{r_{I,X}} = \mathsf{id}_{I \times X} \qquad : I \times X \to I \times X$$

$$\tau^{\mathsf{T}\mathsf{T}'}_{r_{I,X}} = \mathsf{T}\tau^{\mathsf{T}'}_{r_{I,X}} \cdot \tau^{\mathsf{T}}_{r_{\mathsf{T}'I,X}} \qquad : \mathsf{T}\,\mathsf{T}'I \times X \to \mathsf{T}\,\mathsf{T}'(I \times X)$$

and

$$\tau^{\mathsf{T}\times\mathsf{T}'}_{r_{I,X}} = (\tau^{\mathsf{T}}_{r_{I,X}} \times \tau^{\mathsf{T}'}_{r_{I,X}}) \cdot \langle \pi_1 \times \mathsf{id}_X, \pi_2 \times \mathsf{id}_X \rangle$$

$$: (\mathsf{T}I \times \mathsf{T}'I) \times X \to \mathsf{T}(I \times X) \times \mathsf{T}'(I \times X)$$

$$\tau^{\mathsf{T}+\mathsf{T}'}_{r_{I,X}} = (\tau^{\mathsf{T}}_{r_{I,X}} + \tau^{\mathsf{T}'}_{r_{I,X}}) \cdot \mathsf{dl}_{\mathsf{T}I,\mathsf{T}'I,X}$$

$$: (\mathsf{T}I + \mathsf{T}'I) \times X \to \mathsf{T}(I \times X) + \mathsf{T}'(I \times X)$$

Secondly, this result extends to all *regular* functors. In particular, the strength for type functors has been established by [Spe93]. For our purposes here, it is enough, however, to define strength for two extra cases: Id^A and $\mathcal{P}$. We have:

$$\mathsf{T}I^A \times X \xrightarrow{\mathsf{sp}} (\mathsf{T}I^A \times X \times A)^A \xrightarrow{\mathsf{xr}^A} (\mathsf{T}I^A \times A \times X)^A$$

$$\xrightarrow{(\mathsf{ev} \times \mathsf{id})^A} (\mathsf{T}I \times X)^A \xrightarrow{\tau^{\mathsf{T}}_{r_{I,X}}} \mathsf{T}(I \times X)^A$$

and

$$\mathcal{P}\mathsf{T}I \times X \xrightarrow{\tau^{\mathcal{P}}_{r_{\mathsf{T}I,X}}} \mathcal{P}(\mathsf{T}I \times X) \xrightarrow{\mathcal{P}\tau^{\mathsf{T}}_{r_{I,X}}} \mathcal{P}\mathsf{T}(I \times X)$$

where, $\tau^{\mathcal{P}}_{r_{I,X}} : \mathcal{P}I \times X \longrightarrow \mathcal{P}(I \times X)$ is defined in the obvious way:

$$\lambda\langle c, x\rangle \, . \, \{\langle i, x\rangle \mid i \in c\}$$

55. Strong Natural Transformation. A natural transformation $\phi : \mathsf{T} \Longrightarrow \mathsf{R}$ between strong functors is called *strong* iff it verifies

$$\phi \cdot \tau_r^{\mathsf{T}} \;= \tau_r^{\mathsf{R}} \cdot (\phi \times \mathsf{id}) \tag{3.20}$$

It is known that, regarded as natural transformations, the identity, projections, and universal arrows to (respectively, from) the final (respectively, initial) objects in a category are all strong. The same applies to products, splits and eithers of strong natural transformations, as well as to the pre- or post-composition of such a transformation with a strong functor.

56. Strong Unfold. Let T be a strong functor and $\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\ \nu_{\mathsf{T}}$ its final coalgebra. Suppose U is any set and consider $p : U \times C \longrightarrow \mathsf{T}\ U$ a T-*coalgebra with context* C. That is to say, a coalgebra in which the output of the observers depends not only on the carrier U but also on some 'external' context C. To compute the behaviour of p, for different seeds (which are now pairs of state and context), we have to solve uniquely the equation implicit in the following diagram, *i.e.*, to determine the morphism $\mathsf{unfold}_{\mathsf{T}}\, p$ such that the diagram commutes

$$\begin{tikzcd}
\nu_{\mathsf{T}} \arrow[rr, "\omega_{\mathsf{T}}"] & & \mathsf{T}\ \nu_{\mathsf{T}} \\
U \times C \arrow[u, "\mathsf{unfold}_{\mathsf{T}}\, p"] \arrow[r, "\langle p, \pi_2\rangle"] & \mathsf{T}\ U \times C \arrow[r, "\tau_r^{\mathsf{T}}"] & \mathsf{T}\ (U \times C) \arrow[u, "\mathsf{T}\ \mathsf{unfold}_{\mathsf{T}}\, p"']
\end{tikzcd}$$

There are two possible ways to address this question. One is rather trivial: the diagram makes a new T-coalgebra over the product $U \times C$ explicit, and all we have to do is to take the associated anamorphism. So,

$$\mathsf{unfold}_{\mathsf{T}}\, p \;=\; [\![(\tau_r^{\mathsf{T}} \cdot \langle p, \pi_2\rangle)]\!]_{\mathsf{T}} \tag{3.21}$$

The second alternative, although of course equivalent, is conceptually more appealing. Furthermore the 'machinery' involved also applies in the dual case of defining a *strong* *fold* (§61), in which there is not a straightforward answer. The basic idea is to define $\mathsf{unfold}_{\mathsf{T}}\, p$ as an anamorphism for p itself, not in the current category C, but *somewhere else*.

57. Somewhere Else. Consider the category K^{C} whose objects are the same of C, but whose arrows are C-arrows typed as $X \times C \longrightarrow Y$. Given two arrows, say $f : Z \times C \longrightarrow Y$ and $g : X \times C \longrightarrow Z$, their composition is defined as

$$f \bullet g \;=\; f \cdot \langle g, \pi_2 \rangle \tag{3.22}$$

cf.,

$$\begin{tikzcd}
X \times C \arrow[r, "g"] \arrow[dr, "\langle g{,}\pi_2 \rangle"'] & Z \arrow[d, dash, dotted] & \\
 & Z \times C \arrow[r, "f"] & Y
\end{tikzcd}$$

It is easy to check that this form of composition is associative and has $\pi_1 : X \times C \longrightarrow X$ as unit.

We may also define a functor from C to K^{C} to embed our working category in the latter. Clearly, such a functor is the identity on objects, but maps arrows $f : U \longrightarrow Y$ to

$$\dot{\overline{f}} = f \cdot \pi_1 \tag{3.23}$$

This category is, in fact, the Kleisli category for the product *comonad*. Recall that a comonad [BW85] is just the formal dual of a monad, which also finds relevant applications in semantics (see, *e.g.*, [BG92]). However, in the sequel, we will not resort to any specific comonadic property.

58. Coalgebras in K^{C}. Let us go back to the strong functor of §56. There is a 'correspondent' functor on K^{C} given by

$$\dot{\mathsf{T}}\; X \;=\; \mathsf{T}\, X \tag{3.24}$$

$$\dot{\mathsf{T}}\; f \;=\; \mathsf{T}\, f \cdot \tau^{\mathsf{T}}_{r_{X,C}} \tag{3.25}$$

where $f : X \times C \longrightarrow Y$. With a slightly terminological overload, we shall call $\dot{\mathsf{T}}$ the *lifting* of T to K^{C}. Clearly,

$$\dot{\mathsf{T}}\; \dot{\overline{f}} \;=\; \dot{\overline{\mathsf{T}\, f}} \tag{3.26}$$

because

$$\begin{array}{ll}
 & \dot{\overline{\mathsf{T}}}\ \dot{\overline{f}} \\
= & \qquad \{\ \dot{\overline{\mathsf{T}}}\ \text{definition (3.25)}\ \} \\
 & \mathsf{T}\,\dot{\overline{f}} \cdot \tau_r \\
= & \qquad \{\ \text{(3.23)}\ \} \\
 & \mathsf{T}\ (f \cdot \pi_1) \cdot \tau_r \\
= & \qquad \{\ \mathsf{T}\ \text{functor}\ \} \\
 & \mathsf{T}\ f \cdot \mathsf{T}\ \pi_1 \cdot \tau_r \\
= & \qquad \{\ \mathsf{T}\pi_1 \cdot \tau_r = \pi_1\ \text{law (C.12)}\ \} \\
 & \mathsf{T}\ f \cdot \pi_1 \\
= & \qquad \{\ \text{(3.23)}\ \} \\
 & \dot{\overline{\mathsf{T}\,f}}
\end{array}$$

Therefore, a $\dot{\overline{\mathsf{T}}}$-coalgebra is an arrow $p : U \longrightarrow \dot{\overline{\mathsf{T}}}\ U$ in K^{C}, *i.e.*, a C-arrow $p : U \times C \longrightarrow \mathsf{T}\ U$. In the same setting, a $\dot{\overline{\mathsf{T}}}$-comorphism is a C-morphism h making the left diagram to commute:

$$\begin{tikzcd}
U \arrow[r, "p"] \arrow[d, "h"'] & \dot{\overline{\mathsf{T}}}\,U \arrow[d, "\dot{\overline{\mathsf{T}}}\,h"] \\
V \arrow[r, "q"'] & \dot{\overline{\mathsf{T}}}\,V
\end{tikzcd}
\qquad
\begin{tikzcd}
U \times C \arrow[r, "p"] \arrow[d, "h"'] & \mathsf{T}\,U \arrow[r, dotted, no head] & \mathsf{T}\,U \times C \arrow[d, "\dot{\overline{\mathsf{T}}}\,h"] \\
V \arrow[r, dotted, no head] & V \times C \arrow[r, "q"'] & \mathsf{T}\,V
\end{tikzcd}$$

The diagram on the right is the interpretation of the one on the left in the original category. In the sequel, we shall resort to such diagrams as they make the structure involved more explicit. The equation implicit in the diagram is, of course, $\dot{\overline{\mathsf{T}}}\ h \bullet p = q \bullet h$, which corresponds to the usual condition for a coalgebra morphism (§20). Expressed in C it reads: $\mathsf{T}\ h \cdot \tau^{\mathsf{T}}_{r_{U,C}} \cdot \langle p, \pi_2 \rangle = q \cdot \langle h, \pi_2 \rangle$.

59. Final Coalgebras in K^{C}. Is there a final $\dot{\overline{\mathsf{T}}}$-coalgebra? Fortunately the answer is yes and the suitable candidate is just around the corner: $\dot{\overline{\omega_{\mathsf{T}}}}$, *i.e.*, $\omega_{\mathsf{T}} \cdot \pi_1$, which happens to be final for each C in the category of $\dot{\overline{\mathsf{T}}}$-coalgebras. Because of this fact, ω_{T} is called *strongly final*. The justification lies in a theorem asserting that in a *ccc*, every final coalgebra (and, dually, any initial algebra) is *strongly* so (*cf.*, proposition 4.3 in [CS92]).

We can now rephrase the $\mathsf{unfold}_{\mathsf{T}}$ diagram in §56 as an anamorphism diagram in K^{C},

$$\begin{tikzcd}
\nu_{\mathsf{T}} \arrow[r, "\dot{\overline{\omega_{\mathsf{T}}}}"] & \dot{\overline{\mathsf{T}}}\,\nu_{\mathsf{T}} \\
U \arrow[u, "\mathsf{unfold}_{\mathsf{T}}\,p"] \arrow[r, "p"'] & \dot{\overline{\mathsf{T}}}\,U \arrow[u, "\dot{\overline{\mathsf{T}}}\,\mathsf{unfold}_{\mathsf{T}}\,p"']
\end{tikzcd}$$

By finality, $\mathsf{unfold}_{\mathsf{T}}\,p$ is $[\!(p)\!]_{\dot{\overline{\mathsf{T}}}}$. The simple calculation which follows shows this is equivalent to the definition of $\mathsf{unfold}_{\mathsf{T}}\,p$ given in §56.

$$\begin{aligned}
& \dot{\overline{\mathsf{T}}}\,\mathsf{unfold}_{\mathsf{T}}\,p \bullet p \;=\; \dot{\overline{\omega_{\mathsf{T}}}} \bullet \mathsf{unfold}_{\mathsf{T}}\,p \\
\equiv \quad & \{\ \bullet \text{ definition}\ \} \\
& \dot{\overline{\mathsf{T}}}\,\mathsf{unfold}_{\mathsf{T}}\,p \cdot \langle p, \pi_2 \rangle \;=\; \dot{\overline{\omega_{\mathsf{T}}}} \cdot \langle \mathsf{unfold}_{\mathsf{T}}\,p, \pi_2 \rangle \\
\equiv \quad & \{\ \dot{\overline{\mathsf{T}}} \text{ definition}\ \} \\
& \mathsf{T}\,\mathsf{unfold}_{\mathsf{T}}\,p \cdot \tau^{\mathsf{T}}_{r_{U,C}} \cdot \langle p, \pi_2 \rangle \;=\; \dot{\overline{\omega_{\mathsf{T}}}} \cdot \langle \mathsf{unfold}_{\mathsf{T}}\,p, \pi_2 \rangle \\
\equiv \quad & \{\ \omega_{\mathsf{T}} \text{ is strongly final}\ \} \\
& \mathsf{T}\,\mathsf{unfold}_{\mathsf{T}}\,p \cdot \tau^{\mathsf{T}}_{r_{U,C}} \cdot \langle p, \pi_2 \rangle \;=\; \omega_{\mathsf{T}} \cdot \pi_1 \cdot \langle \mathsf{unfold}_{\mathsf{T}}\,p, \pi_2 \rangle \\
\equiv \quad & \{\ \times \text{ cancellation}\ \} \\
& \mathsf{T}\,\mathsf{unfold}_{\mathsf{T}}\,p \cdot \tau^{\mathsf{T}}_{r_{U,C}} \cdot \langle p, \pi_2 \rangle \;=\; \omega_{\mathsf{T}} \cdot \mathsf{unfold}_{\mathsf{T}}\,p
\end{aligned}$$

60. LAWS. We are now ready to state the universal property of unfolds for a strong type,

$$k = \mathsf{unfold}_{\mathsf{T}}\,p \;\Leftrightarrow\; \dot{\overline{\omega_{\mathsf{T}}}} \bullet k = \dot{\overline{\mathsf{T}}}\,k \bullet p \tag{3.27}$$

from which the following *cancellation* and *reflection* results are easily derived:

$$\dot{\overline{\omega_{\mathsf{T}}}} \bullet \mathsf{unfold}_{\mathsf{T}}\, p = \dot{\overline{\mathsf{T}}}\; \mathsf{unfold}_{\mathsf{T}}\, p \bullet p \tag{3.28}$$

$$\mathsf{unfold}_{\mathsf{T}}\, \dot{\overline{\omega_{\mathsf{T}}}} = \mathsf{id}_{\nu_{\mathsf{T}}} \tag{3.29}$$

Moreover, for two coalgebras p and q and a coalgebra morphism f between them (*i.e.*, such that $p \bullet f = \dot{\overline{\mathsf{T}}}\; f \bullet q$), one gets the following *fusion* law for unfolds:

$$\mathsf{unfold}_{\mathsf{T}}\, p \bullet f \;=\; \mathsf{unfold}_{\mathsf{T}}\, q \tag{3.30}$$

To conclude, we prove that, for a 'normal' coalgebra $p : U \longrightarrow \mathsf{T}\, U$, $\mathsf{unfold}_{\mathsf{T}}\, \dot{\overline{p}}$ is the lifting to K^{C} of the corresponding anamorphism.

Proof. Replace k by $\dot{\overline{[\![(p)]\!]_{\mathsf{T}}}}$ in (3.27). Then,

$$\begin{aligned}
& \dot{\overline{\mathsf{T}}}\; \dot{\overline{[\![(p)]\!]_{\mathsf{T}}}} \bullet \dot{\overline{p}} \\
= & \qquad \{\ (3.26)\ \} \\
& \dot{\overline{\mathsf{T}\; [\![(p)]\!]_{\mathsf{T}}}} \bullet \dot{\overline{p}} \\
= & \qquad \{\ \text{functor}\ \} \\
& \dot{\overline{\mathsf{T}\; [\![(p)]\!]_{\mathsf{T}} \cdot p}} \\
= & \qquad \{\ \text{ana universal (3.6)}\ \} \\
& \dot{\overline{\omega_{\mathsf{T}} \cdot [\![(p)]\!]_{\mathsf{T}}}} \\
= & \qquad \{\ \text{functor}\ \} \\
& \dot{\overline{\omega_{\mathsf{T}}}} \bullet \dot{\overline{[\![(p)]\!]_{\mathsf{T}}}}
\end{aligned}$$

□

61. STRONG FOLD. A dual construction gives the definition of a *strong fold*. That is to say, a strong fold is just a catamorphism in K^{C}. To see this first recall that, in a *ccc*, an initial algebra is also strongly initial. Therefore, consider $m : \mathsf{T}\ D \times C \longrightarrow D$ a T-*algebra over D, with context C*, which amounts to a $\dot{\overline{\mathsf{T}}}$-algebra in K^{C}, and define $\mathsf{fold}_{\mathsf{T}}\, m$ as the induced catamorphism there. As usual, this is the unique arrow which makes the following diagram to commute,

$$\begin{tikzcd}
\dot{\overline{\mathsf{T}}}\,\mu_{\mathsf{T}} \arrow[r, "\dot{\overline{\alpha_{\mathsf{T}}}}"] \arrow[d, "\dot{\overline{\mathsf{T}}}\,\mathsf{fold}_{\mathsf{T}}\,m"'] & \mu_{\mathsf{T}} \arrow[d, "\mathsf{fold}_{\mathsf{T}}\,m"] \\
\dot{\overline{\mathsf{T}}}\,D \arrow[r, "m"'] & D
\end{tikzcd}$$

where $\dot{\overline{\alpha_{\mathsf{T}}}}$ is, as before, $\alpha_{\mathsf{T}} \cdot \pi_1$. We may now rephrase the equation implicit in the diagram in the context of C:

$$\mathsf{fold}_{\mathsf{T}}\,m \bullet \dot{\overline{\alpha_{\mathsf{T}}}} \;=\; m \bullet \dot{\overline{\mathsf{T}}}\,\mathsf{fold}_{\mathsf{T}}\,m$$

$\equiv$ $\{\ \bullet$ definition $\}$

$$\mathsf{fold}_{\mathsf{T}}\,m \cdot \langle \dot{\overline{\alpha_{\mathsf{T}}}}, \pi_2 \rangle \;=\; m \cdot \langle \dot{\overline{\mathsf{T}}}\,\mathsf{fold}_{\mathsf{T}}\,m, \pi_2 \rangle$$

$\equiv$ $\{\ \alpha_{\mathsf{T}}$ is strongly initial, $\dot{\overline{\mathsf{T}}}$ definition $\}$

$$\mathsf{fold}_{\mathsf{T}}\,m \cdot \langle \alpha_{\mathsf{T}} \cdot \pi_1, \pi_2 \rangle \;=\; m \cdot \langle \mathsf{T}\ \mathsf{fold}_{\mathsf{T}}\,m \cdot \tau^{\mathsf{T}}_{r_{D,C}}, \pi_2 \rangle$$

$\equiv$ $\{\ \times$ absorption $\}$

$$\mathsf{fold}_{\mathsf{T}}\,m \cdot (\alpha_{\mathsf{T}} \times \mathsf{id}) \cdot \langle \pi_1, \pi_2 \rangle \;=\; m \cdot \langle \mathsf{T}\ \mathsf{fold}_{\mathsf{T}}\,m \cdot \tau^{\mathsf{T}}_{r_{D,C}}, \pi_2 \rangle$$

$\equiv$ $\{\ \times$ reflection $\}$

$$\mathsf{fold}_{\mathsf{T}}\,m \cdot (\alpha_{\mathsf{T}} \times \mathsf{id}) \;=\; m \cdot \langle \mathsf{T}\ \mathsf{fold}_{\mathsf{T}}\,m \cdot \tau^{\mathsf{T}}_{r_{D,C}}, \pi_2 \rangle$$

We have, thus, defined the universal property for this functional:

$$k = \mathsf{fold}_{\mathsf{T}}\,p \;\Leftrightarrow\; k \bullet \dot{\overline{\alpha_{\mathsf{T}}}} = m \bullet \dot{\overline{\mathsf{T}}}\,k \tag{3.31}$$

As one would expect, all laws in §60 have a dual version here. The corresponding *cancellation*, *reflection* and *fusion* laws are, respectively,

$$\mathsf{fold}_{\mathsf{T}}\,m \bullet \dot{\overline{\alpha_{\mathsf{T}}}} \;=\; m \bullet \dot{\overline{\mathsf{T}}}\,\mathsf{fold}_{\mathsf{T}}\,m \tag{3.32}$$

$$\mathsf{fold}_{\mathsf{T}}\,\dot{\overline{\alpha_{\mathsf{T}}}} \;=\; \mathsf{id}_{\mu_{\mathsf{T}}} \tag{3.33}$$

$$f \bullet m = n \bullet \dot{\overline{\mathsf{T}}}\,m \;\Rightarrow\; f \bullet \mathsf{fold}_{\mathsf{T}}\,p \;=\; \mathsf{fold}_{\mathsf{T}}\,n \tag{3.34}$$

Furthermore, for a 'normal' algebra $m : \mathsf{T}\ D \longrightarrow D$, $\mathsf{fold}_{\mathsf{T}}\,\dot{\overline{m}}$ corresponds exactly to the lifting to K^{C} of the corresponding catamorphism, *i.e.*, to $\dot{\overline{(\![m]\!)_{\mathsf{T}}}}$. Notice, however, that $\mathsf{fold}_{\mathsf{T}}\,\dot{\overline{m}}$, for the general case, cannot be expressed as a catamorphism in C, which makes this shift to 'somewhere else' necessary to characterise it.

62. STRONG MAP. The action of a functor T on morphisms 'with context', *i.e.*, morphisms $f : X \times C \longrightarrow Y$, is defined as the correspondent action of the lifting of T on f seen as a K^{C} morphism. For notational reasons, and to emphasise the correspondence with the CHARITY (abstract) combinators' notation [CS95], this will be denoted by $\mathsf{map}_{\mathsf{T}}\, f$. Therefore, define

$$\mathsf{map}_{\mathsf{T}}\, f \;=\; \dot{\overline{\overline{\mathsf{T}}}}\, f \;=\; \mathsf{T}\, X \times C \xrightarrow{{\tau_r}^{\mathsf{T}}_{X,C}} \mathsf{T}\,(X \times C) \xrightarrow{\mathsf{T}\, f} \mathsf{T}\, Y$$

In order to 'complete the picture', we shall now give the strong versions of type and cotype functors.

63. COTYPE FUNCTORS REVISITED. The fact that ω_{T} (resp., α_{T}), in a *ccc*, is strongly final (resp., initial), opens the possibility of defining strong cotype (resp., type) functors. We shall denote them by $\underline{\mathsf{N}}_{\mathsf{T}}$ (resp., $\underline{\mathsf{M}}_{\mathsf{T}}$). Their action on objects is, as expected,

$$\begin{aligned} \underline{\mathsf{N}}_{\mathsf{T}}\, A &= \nu_{\mathsf{T}_A} \\ \underline{\mathsf{M}}_{\mathsf{T}}\, A &= \mu_{\mathsf{T}_A} \end{aligned}$$

Let us see how to define the action of $\underline{\mathsf{N}}_{\mathsf{T}}$ on a morphism $f : A \times C \longrightarrow B$. The construction is similar to the one used in the definition of standard cotype functors (§47). As depicted in the diagram below, in a strong setting it is defined as an *unfold*:

$$\begin{tikzcd}
\underline{\mathsf{N}}_{\mathsf{T}}\, B \arrow[r, dash, dotted] & \underline{\mathsf{N}}_{\mathsf{T}}\, B \times C \arrow[rrrr, "\dot{\overline{\omega_{\mathsf{T}_B}}}"] & & & & \mathsf{T}_B\, \underline{\mathsf{N}}_{\mathsf{T}}\, B \\
\underline{\mathsf{N}}_{\mathsf{T}}\, A \times C \arrow[u, "\underline{\mathsf{N}}_{\mathsf{T}}\, f"] \arrow[r, "\dot{\overline{\omega_{\mathsf{T}_A}}}"] & \mathsf{T}_A\, \underline{\mathsf{N}}_{\mathsf{T}}\, A \arrow[r, dash, dotted] & \mathsf{T}_A\, \underline{\mathsf{N}}_{\mathsf{T}}\, A \times C \arrow[rr, "\dot{\overline{\overline{\mathsf{T}}}}\,(\mathsf{id}_{\underline{\mathsf{N}}_{\mathsf{T}}\, A}, f)"] & & \mathsf{T}_B\, \underline{\mathsf{N}}_{\mathsf{T}}\, A \arrow[r, dash, dotted] & \mathsf{T}_B\, \underline{\mathsf{N}}_{\mathsf{T}}\, A \times C \arrow[u, "\mathsf{T}_B\, \underline{\mathsf{N}}_{\mathsf{T}}\, f"']
\end{tikzcd}$$

$\underline{\mathsf{N}}_{\mathsf{T}}\, f$ is, then, defined as the arrow making the diagram to commute, *i.e.*,

$$\dot{\overline{\omega_{\mathsf{T}_B}}} \bullet \underline{\mathsf{N}}_{\mathsf{T}}\, f \;=\; \mathsf{T}_B\, \underline{\mathsf{N}}_{\mathsf{T}}\, f \bullet \dot{\overline{\overline{\mathsf{T}}}}\,(\mathsf{id}_{\underline{\mathsf{N}}_{\mathsf{T}}\, A}, f) \bullet \dot{\overline{\omega_{\mathsf{T}_A}}}$$

which is expressed in terms of C-arrows as,

$$\begin{aligned}
& \dot{\overline{\omega_{\mathsf{T}_B}}} \bullet \underline{\mathsf{N}}_{\mathsf{T}}\, f \;=\; \mathsf{T}_B\, \underline{\mathsf{N}}_{\mathsf{T}}\, f \bullet \dot{\overline{\mathsf{T}}}\,(\mathsf{id}_{\underline{\mathsf{N}}_{\mathsf{T}}\, A}, f) \bullet \dot{\overline{\omega_{\mathsf{T}_A}}} \\
\equiv \quad & \{\ \bullet, \dot{\overline{\omega_{\mathsf{T}_B}}} \text{ definition } \} \\
& \omega_{\mathsf{T}_B} \cdot \pi_1 \cdot \langle \underline{\mathsf{N}}_{\mathsf{T}}\, f, \pi_2 \rangle \;=\; \dot{\overline{\mathsf{T}}}\,(\underline{\mathsf{N}}_{\mathsf{T}}\, f, f) \bullet \dot{\overline{\omega_{\mathsf{T}_A}}} \\
\equiv \quad & \{\ \bullet, \dot{\overline{\mathsf{T}_A}} \text{ definition and } \times \text{ cancellation } \} \\
& \omega_{\mathsf{T}_B} \cdot \underline{\mathsf{N}}_{\mathsf{T}}\, f \;=\; \dot{\overline{\mathsf{T}}}\,(\underline{\mathsf{N}}_{\mathsf{T}}\, f, f) \cdot \langle \omega_{\mathsf{T}_A} \cdot \pi_1, \pi_2 \rangle \\
\equiv \quad & \{\ \times \text{ absorption}, \times \text{ reflection } \} \\
& \omega_{\mathsf{T}_B} \cdot \underline{\mathsf{N}}_{\mathsf{T}}\, f \;=\; \dot{\overline{\mathsf{T}}}\,(\underline{\mathsf{N}}_{\mathsf{T}}\, f, f) \cdot (\omega_{\mathsf{T}_A} \times \mathsf{id}_C)
\end{aligned}$$

As this is an unfold diagram, we have

$$\underline{\mathsf{N}}_{\mathsf{T}}\, f \;=\; \mathsf{unfold}_{\mathsf{T}_B}\, (\dot{\overline{\mathsf{T}}}\,(\mathsf{id}_{\underline{\mathsf{N}}_{\mathsf{T}}\, A}, f) \bullet \dot{\overline{\omega_{\mathsf{T}_A}}}) \tag{3.35}$$

Now, notice that the lifting of the standard cotype functor applied to the same f yields arrow

$$\mathsf{map}_{\mathsf{N}_{\mathsf{T}}}\, f \;=\; \dot{\overline{\mathsf{N}_{\mathsf{T}}}}\, f \;=\; \mathsf{N}_{\mathsf{T}}\, f \cdot \tau_r^{\mathsf{N}_{\mathsf{T}}} \tag{3.36}$$

Naturally, we would like to see the strong cotype functor as the lifting to K^{C} of the 'standard' one. The strength $\tau_r^{\mathsf{N}_{\mathsf{T}}}$ for the cotype functor can actually be defined such that $\underline{\mathsf{N}}_{\mathsf{T}}\, f = \mathsf{N}_{\mathsf{T}}\, f \cdot \tau_r^{\mathsf{N}_{\mathsf{T}}}$. This is proved for the (dual) type functor case, in [Par96].

Using the map notation, the definition of $\underline{\mathsf{N}}_{\mathsf{T}}\, f$ looks like

$$\mathsf{map}_{\mathsf{N}_{\mathsf{T}}}\, f \;=\; \mathsf{unfold}_{\mathsf{T}_B}\, (\mathsf{map}_{\mathsf{T}}\, (\mathsf{map}_{\mathsf{N}_{\mathsf{T}}}\, f, f) \bullet \dot{\overline{\omega_{\mathsf{T}_A}}})$$

which, from a programming point of view, would serve as the definition of the map combinator for coinductive types as a functional in a programming language.

64. TYPE FUNCTORS REVISITED. A dual construction is used to define the action on morphisms of the strong type functor. As expected, it arises as a fold. The actual definition is

$$\underline{\mathsf{M}}_{\mathsf{T}}\, f \;=\; \mathsf{fold}_{\mathsf{T}_A}\, \dot{\overline{\alpha_{\mathsf{T}_B}}} \bullet \dot{\overline{\mathsf{T}}}\,(\mathsf{id}_{\underline{\mathsf{M}}_{\mathsf{T}}\, B}, f)$$

as follows from the diagram below.

$$\begin{tikzcd}
\mathsf{T}_A\, \underline{\mathsf{M}}_{\mathsf{T}}\, A \times C \arrow[rrr, "\dot{\overline{\alpha_{\mathsf{T}_A}}}"] \arrow[d, "\mathsf{T}_A\, \underline{\mathsf{M}}_{\mathsf{T}}\, f"'] & & & \underline{\mathsf{M}}_{\mathsf{T}}\, A \arrow[r, dash, dotted] & \underline{\mathsf{M}}_{\mathsf{T}}\, A \times C \arrow[d, "\underline{\mathsf{M}}_{\mathsf{T}}\, f"] \\
\mathsf{T}_A\, \underline{\mathsf{M}}_{\mathsf{T}}\, B \arrow[r, dash, dotted] & \mathsf{T}_A\, \underline{\mathsf{M}}_{\mathsf{T}}\, B \times C \arrow[r, "\dot{\mathsf{T}}\, (\mathsf{id}_{\underline{\mathsf{M}}_{\mathsf{T}}\, B}, f)"] & \mathsf{T}_B\, \underline{\mathsf{M}}_{\mathsf{T}}\, B \arrow[r, dash, dotted] & \mathsf{T}_B\, \underline{\mathsf{M}}_{\mathsf{T}}\, B \times C \arrow[r, "\dot{\overline{\alpha_{\mathsf{T}_B}}}"] & \underline{\mathsf{M}}_{\mathsf{T}}\, B
\end{tikzcd}$$

Again we may use the map notation and write

$$\mathsf{map}_{\mathsf{M}_{\mathsf{T}}}\, f \;=\; \mathsf{fold}_{\mathsf{T}_A}\, (\dot{\overline{\alpha_{\mathsf{T}_B}}} \bullet \mathsf{map}_{\mathsf{T}}\, (\mathsf{map}_{\mathsf{M}_{\mathsf{T}}}\, f, f))$$

as a definition of the map combinator for inductive types.

CHAPTER 4

Coalgebraic Models for Processes and Components

Summary

This chapter develops coalgebraic models of both processes and software components. Processes are introduced as inhabitants of the carriers of final coalgebras upon which the usual process combinators are defined. This leads to a 'reconstruction' of a basic process calculus along the lines of the 'Bird-Meertens' formalism. In particular, process properties are proved in an equational, essentially pointfree, way and the constructions put forward are directly prototyped in a functional language supporting coinductive types. If processes, as 'pure' behaviours, are elements of final coalgebras, just as sequences or trees are elements of initial algebras, software components have to deal with both behaviour and data, in the form of input/output and state information. They are thus described as concrete coalgebras, with specified initial conditions and parametric on a behaviour captured by a strong monad.

1. Shapes

1. PROCESSES AND COMPONENTS. This chapter discusses both *processes* and *components* from a coalgebraic point of view. The former, as 'pure' behaviours are studied as elements of coinductive data types, *i.e.*, inhabitants of (the carrier of) a *final* coalgebra. The latter are regarded as state-based interactive systems and modelled as concrete coalgebras.

After this introductory section on coalgebraic modelling, sections 2, 3 and 4 are entirely devoted to the development of a family of CCS-like process calculi along the lines of the so-called *Bird-Meertens formalism* [Bir87, BM87] and their animation in CHARITY. Finally, section 5 introduces the component models to be used in the following chapters. Their parameterization by a strong monad to capture particular behaviour models is emphasised.

Before that, however, we shall go through a small 'warming up' exercise to investigate some elementary shapes for dynamic systems. It turns out that very simple functors are enough to model a variety of such systems common in, *e.g.*, process calculi or automaton theory. For each Set-endofunctor T, we shall ask ourselves how T-coalgebras look like and what the appropriate notions of comorphism and bisimulation are. This will unveil what should be understood by behavioural equivalence in each case. We shall also take the opportunity to introduce the notion of an *invariant* wrt a coalgebraic structure and refer briefly (*i.e.*, as far as such concepts are needed in later sections) recent work by B. Jacobs and others on associated modal languages.

2. SHAPES. In modelling data structures as T-algebras, the choice of functor T fixes the associated syntactic information — T is, in fact, an abstract description of the structure's signature (or 'abstract grammar'). Such a choice is equally relevant when regarding dynamical systems as coalgebras. In this case, T captures both a signature of actions and observers — the system's *interface* — as well as a particular behavioural model. These two aspects are orthogonal and, as argued in the sequel, should be dealt separately as far as possible.

The table below suggests a possible taxonomy of functors for dynamic systems. The second column consists of functors whose coalgebras are models of deterministic systems. The third column introduces nondeterminism, as usually considered in, *e.g.*, process calculi or automaton theory. Rows, on the other hand, classify systems by the interface they exhibit. The degenerated cases simply lack any accessible interface. Reactive systems accept external stimuli, of type I, which trigger their evolution. Active systems, on the other hand, provide a state observer of type O and an action whose occurrence does not depend on any external stimulus.

	Deterministic Systems	Non Deterministic Systems
Degenerated	$\mathsf{T}\, X \;=\; X$ (1)	$\mathsf{T}\, X \;=\; \mathcal{P}X$ (4)
Active	$\mathsf{T}\, X \;=\; O \times X$ (2)	$\mathsf{T}\, X \;=\; O \times \mathcal{P}X$ (5) $\mathsf{T}\, X \;=\; \mathcal{P}(O \times X)$ (5')
Reactive	$\mathsf{T}\, X \;=\; X^I$ (3)	$\mathsf{T}\, X \;=\; \mathcal{P}(X)^I$ (6)

For each case, let us consider two T-coalgebras, p and q, over carriers U and V, respectively, and investigate what comorphisms and bisimulation are, recalling §3.20 and §3.23:

$$\begin{array}{rcl} h : p \longrightarrow q \text{ is a comorphism} & \equiv & \mathsf{T}\, h \cdot p \;=\; q \cdot h \\ \langle R, \rho \rangle \text{ is a bisimulation on } p \text{ and } q & \equiv & \mathsf{T}\, \pi_1 \cdot \rho = p \cdot \pi_1 \;\wedge\; \mathsf{T}\, \pi_2 \cdot \rho = q \cdot \pi_2 \end{array}$$

CASE 1. For the identity functor, coalgebras simply are functions with the same source and target, *i.e.*, carrier U. Moreover, any transition respecting relation is a bisimulation, because

$$\begin{array}{ll} & \pi_1 \cdot \rho = p \cdot \pi_1 \;\wedge\; \pi_2 \cdot \rho = q \cdot \pi_2 \\ \equiv & \qquad \{\; \times \text{ equality} \;\} \\ & \langle \pi_1 \cdot \rho, \pi_2 \cdot \rho \rangle \;=\; \langle p \cdot \pi_1, q \cdot \pi_2 \rangle \\ \equiv & \qquad \{\; \times \text{ fusion, } \times \text{ absorption} \;\} \\ & \langle \pi_1, \pi_2 \rangle \cdot \rho \;=\; p \times q \cdot \langle \pi_1, \pi_2 \rangle \\ \equiv & \qquad \{\; \times \text{ reflection} \} \\ & \rho \;=\; p \times q \end{array}$$

The final coalgebra is $\langle \mathbf{1}, \mathsf{id}_{\mathbf{1}} \rangle$.

CASE 2. Coalgebra dynamics for this functor are splits $\langle o_p, a_p\rangle$ of an *attribute* $o_p : U \longrightarrow O$ and an *action* $a_p : U \longrightarrow U$. An arbitrary coalgebra can thus be written as $p = \langle U, \langle o_p, a_p\rangle\rangle$. The condition on comorphisms yields

$$\begin{array}{ll}
 & (\mathsf{id} \times h) \cdot \langle o_p, a_p\rangle \;=\; \langle o_q, a_q\rangle \cdot h \\
\equiv & \qquad \{ \text{ × absorption } \} \\
 & \langle o_p, h \cdot a_p\rangle \;=\; \langle o_q \cdot h, a_q \cdot h\rangle \\
\equiv & \qquad \{ \text{ going pointwise } \} \\
 & o_p\ u = (o_q \cdot h)\ u \ \wedge\ (h \cdot a_p)\ u = (a_q \cdot h)\ u
\end{array}$$

Similarly, the proof rule for bisimulation becomes

$$\begin{array}{ll}
 & (\mathsf{id} \times \pi_1) \cdot \langle o_\rho, a_\rho\rangle = \langle o_p, a_p\rangle \cdot \pi_1 \ \wedge\ (\mathsf{id} \times \pi_2) \cdot \langle o_\rho, a_\rho\rangle = \langle o_q, a_q\rangle \cdot \pi_2 \\
\equiv & \qquad \{ \text{ × absorption, × fusion } \} \\
 & \langle o_\rho, \pi_1 \cdot a_\rho\rangle = \langle o_p \cdot \pi_1, a_p \cdot \pi_1\rangle \ \wedge\ \langle o_\rho, \pi_2 \cdot a_\rho\rangle = \langle o_q \cdot \pi_2, a_q \cdot \pi_2\rangle \\
\equiv & \qquad \{ \text{ rearranging } \} \\
 & o_\rho = o_p \cdot \pi_1 = o_q \cdot \pi_2 \ \wedge\ \langle \pi_1 \cdot a_\rho, \pi_2 \cdot a_\rho\rangle = \langle a_p \cdot \pi_1, a_q \cdot \pi_2\rangle \\
\equiv & \qquad \{ \text{ × fusion, × absorption} \} \\
 & o_\rho = o_p \cdot \pi_1 = o_q \cdot \pi_2 \ \wedge\ \langle \pi_1, \pi_2\rangle \cdot a_\rho = (a_p \times a_q) \cdot \langle \pi_1, \pi_2\rangle \\
\equiv & \qquad \{ \text{ × reflection } \} \\
 & o_\rho = o_p \cdot \pi_1 = o_q \cdot \pi_2 \ \wedge\ a_\rho = a_p \times a_q \\
\equiv & \qquad \{ \text{ going pointwise } \} \\
 & \langle u, v\rangle \in R \ \Rightarrow\ o_p\ u = o_q\ v \ \wedge\ \langle a_p\ u, a_q\ v\rangle \in R
\end{array}$$

In this case, the final coalgebra is well-known: its carrier is O^ω, the set of O-streams and its dynamics is given by the split of the 'head' and the 'tail' functions over them.

CASE 3. Coalgebra dynamics for this functor can be expressed by a function $p : U \times I \longrightarrow U$ in curried form. The bisimulation rule becomes

$$\begin{array}{ll}
 & \pi_1^I \cdot \overline{\rho} = \overline{p} \cdot \pi_1 \ \wedge\ \pi_2^I \cdot \overline{\rho} = \overline{q} \cdot \pi_1 \\
\equiv & \qquad \{ \text{ exponential absorption } \} \\
 & \overline{\pi_1 \cdot \rho} = \overline{p} \cdot \pi_1 \ \wedge\ \overline{\pi_2 \cdot \rho} = \overline{q} \cdot \pi_2 \\
\equiv & \qquad \{ \text{ exponential fusion } \} \\
 & \overline{\pi_1 \cdot \rho} = \overline{p \cdot (\pi_1 \times \mathsf{id})} \ \wedge\ \overline{\pi_2 \cdot \rho} = \overline{q \cdot (\pi_2 \times \mathsf{id})}
\end{array}$$

$$
\begin{array}{ll}
\equiv & \quad \{ \text{ exponential universal } \} \\
& \pi_1 \cdot \rho = p \cdot (\pi_1 \times \mathsf{id}) \ \wedge \ \pi_2 \cdot \rho = q \cdot (\pi_2 \times \mathsf{id}) \\
\equiv & \quad \{ \ \times \text{ universal } \} \\
& \rho \ = \ \langle p \cdot (\pi_1 \times \mathsf{id}), q \cdot (\pi_2 \times \mathsf{id}) \rangle \\
\equiv & \quad \{ \text{ going pointwise } \} \\
& \langle u, v \rangle \in R \ \Rightarrow \ \forall_{i \in I} \ . \ \langle p \ \langle u, i \rangle, q \ \langle v, i \rangle \rangle \in R
\end{array}
$$

The entries in the second column of the table are obtained from the corresponding ones in the first column by composing with the finite powerset functor, which captures nondeterminism as a (particular) behavioural effect.

CASE 4. Coalgebra dynamics are functions from U to the finite powerset of U, which are isomorphic to (image finite) binary relations on U. The comorphism condition yields the following equality

$$
\begin{array}{ll}
& \mathcal{P}h \cdot p \ = \ q \cdot h \\
\equiv & \quad \{ \text{ going pointwise } \} \\
& x \in \mathcal{P}h \ (p\, u) \ \ \mathsf{iff} \ \ x \in q \ (h\, u) \\
\equiv & \quad \{ \text{ equivalence} \} \\
& (y \in p\ u \Rightarrow \ h\ y \in q\ (h\ u) \ \ \wedge \ \ (x \in q\ (h\, u) \ \Rightarrow \ \exists_{y \in p\, u} \ . \ x = h\ y)
\end{array}
$$

The first implication is a *preservation* condition which corresponds to the usual characterisation of a morphism between relations or transition systems (see, *e.g.*, [WN95]). However, the notion of a comorphism is much stronger as it also entails the second implication. Thinking of a relation r as a (nondeterministic) transition system given by

$$u \longrightarrow v \ \ \mathsf{iff} \ \ \langle u, v \rangle \in r$$

this second implication means that transitions are also *reflected* backwards.

In general, the commutativity of the square defining a comorphism gives rise to two implications which capture, not only the preservation of the coalgebra dynamics, but also its reflection. This may give some intuition on why comorphisms entail bisimulation. For this particular case, a relation R is a bisimulation iff

$$\langle u, v \rangle \in R \ \Rightarrow \ \forall_{x \in p\, u} \ . \ \exists_{y \in q\, v} \ . \ \langle x, y \rangle \in R \ \wedge \ \forall_{y \in q\, v} \ . \ \exists_{x \in p\, u} \ . \ \langle x, y \rangle \in R$$

CASES 5 AND 6. These remaining cases amount to combinations of (4) with (2) and (3), respectively. Notice, in particular, that (5) and (5') witness two possible compositions of functors (2) and (4), both having meaningful computational interpretations.

Case (5) models objects able to evolve silently in a nondeterministic way while being observed in O. The bisimulation rule is

$$\begin{aligned}\langle u, v\rangle \in R \;\Rightarrow\; & o_p\; u = o_q\; v \wedge \\ & \forall_{x \in a_p\; u} \,.\, \exists_{y \in a_q\; v} \,.\, \langle x, y\rangle \in R \;\wedge\; \forall_{y \in a_q\; v} \,.\, \exists_{x \in a_p\; u} \,.\, \langle x, y\rangle \in R\end{aligned}$$

Systems with shape (5'), on the other hand, evolve nondeterministically producing a visible effect of type O. As argued in the next section, this captures a possible shape for (image finite) CCS-like processes. Bisimulation in this case corresponds to what is called strict bisimulation in the CCS literature:

$$\langle u, v\rangle \in R \Rightarrow \forall_{\langle o,x\rangle \in p\, u} \,.\, \exists_{\langle o,y\rangle \in q\, v} \,.\, \langle x, y\rangle \in R \wedge \forall_{\langle o,y\rangle \in q\, v} \,.\, \exists_{\langle o,x\rangle \in p\, u} \,.\, \langle x, y\rangle \in R$$

Case (6) is the corresponding reactive version.

3. TRANSITIONS. Just as any transition system can be coded back as a coalgebra, any coalgebra $\langle U, p : U \longrightarrow \mathsf{T}U\rangle$ specifies a (T-shaped) transition structure over its carrier U. For extended polynomial Set endofunctors (§3.10) such a structure may be expressed as a binary relation $\longrightarrow_p \subseteq U \times U$, defined in terms of the *structural membership* relation $\in_{\mathsf{T}} \subseteq U \times \mathsf{T}\, U$, *i.e.*,

$$u \longrightarrow_p u' \quad \mathsf{iff} \quad u' \in_{\mathsf{T}} p\; u$$

where $\in_{\mathsf{T}}$ is defined by induction of the structure of T:

$$\begin{array}{rcl}
x \in_{\mathsf{Id}} y & \mathsf{iff} & x = y \\
x \in_{\underline{K}} y & \mathsf{iff} & \mathsf{false} \\
x \in_{\mathsf{T}_1 \times \mathsf{T}_2} y & \mathsf{iff} & x \in_{\mathsf{T}_1} \pi_1\; y \;\vee\; x \in_{\mathsf{T}_2} \pi_2\; y \\
x \in_{\mathsf{T}_1 + \mathsf{T}_2} y & \mathsf{iff} & \begin{cases} y = \iota_1\; y' & \Rightarrow x \in_{\mathsf{T}_1} y' \\ y = \iota_2\; y' & \Rightarrow x \in_{\mathsf{T}_2} y' \end{cases} \\
x \in_{\mathsf{T}^{\underline{K}}} y & \mathsf{iff} & \exists_{k \in K} .\; x \in_{\mathsf{T}} y\; k \\
x \in_{\mathcal{P}\mathsf{T}} y & \mathsf{iff} & \exists_{y' \in y} .\; x \in_{\mathsf{T}} y'
\end{array}$$

4. EXAMPLES. Referring again to the table in §2, the transition relations associated to cases (1) and (4) are, respectively,

$$u \longrightarrow_p u' \quad \mathsf{iff} \quad u' = p\; u$$

and

$$u \longrightarrow_p u' \quad \text{iff} \quad u' \in p\ u$$

Cases (5) and (6) yield, respectively,

$$u \longrightarrow_p u' \quad \text{iff} \quad u' \in \pi_2\ (p\ u)$$

and

$$u \longrightarrow_p u' \quad \text{iff} \quad \exists_{i \in I}.\ u' \in (p\ u)\ i$$

In some cases it is convenient to express the coalgebra dynamics as a family of binary relations indexed by the 'interface' parameters involved in the definition of T. Case (5), for example, could give rise to

$$u \overset{o}{\longrightarrow}_p u' \quad \text{iff} \quad o = \pi_1\ u\ \wedge\ u' \in \pi_2\ u$$

and case (5') to

$$u \overset{o}{\longrightarrow}_p u' \quad \text{iff} \quad \langle o, u' \rangle \in u$$

which corresponds to the transition relation generated by structural membership regarding T as a *bifunctor* (in X and O).

5. Invariants. A predicate $\phi : U \longrightarrow \mathbf{2}$ over the carrier of a T-coalgebra $\langle U, p : U \longrightarrow \mathsf{T}\ U \rangle$ is said to be a *p-invariant* if it is closed under the coalgebra dynamics. This may be formalised as follows. First define, for ϕ such a predicate, combinator $\bigcirc_p$:

$$(\bigcirc_p\ \phi)\ u \ \text{ iff } \ \forall_{u' \in_{\mathsf{T}} p\ u}\ .\ \phi\ u'$$

which means that $\bigcirc_p\ \phi$ holds in all states whose immediate successor states (if any), under coalgebra p, if any, satisfy ϕ. Then ϕ is an invariant iff

$$\phi\ \Rightarrow\ \bigcirc_p\ \phi$$

6. Modal Languages. Combinator $\bigcirc_p$ above can be regarded as a *modal* combinator which corresponds to the familiar (weak) *next* operator in modal logics. The key observation is that, in this case, the structure upon which the operator is interpreted is the transition system defined by the coalgebra p. In fact, it has been recently recognised by a number of authors (notably in [Mos99] and [Jac99b]) that a modal language associated to a coalgebra p is determined by its *shape*, as recorded by the relevant functor. As one could expect, the topic is a 'big issue' in coalgebra theory. As usual, we shall however restrict ourselves to a brief introduction to Jacobs' approach [Jac99b] in order to characterise a few notions used in the sequel. We shall also consider only coalgebras in Set, although most results can be carried to more general contexts.

The above mentioned *next* operator is introduced in [Jac99b] in a slightly different, but equivalent, way. The basic machinery is the extension of predicates, considered themselves as sets, along (extended) polynomial functors, which appears in a very general setting in [HJ98]. The extension is computed by a function $(\ )^{\mathsf{T}} : \mathcal{P}X \longrightarrow \mathcal{P}\mathsf{T}\ X$ defined inductively on the structure of functor T. Of course, $(\phi)^{\mathsf{T}}$ coincides with $\mathsf{T}\ \phi$, considering ϕ as a set itself. The interest of the definition below lies in providing a convenient way to deal with the different cases. Moreover, it allows the explicit description of a left adjoint $(\ )_{\mathsf{T}}$ which, 'unlifting' the predicate, paves the way to the introduction of 'past' modalities. A concrete description of $(\phi)^{\mathsf{T}}$ and $(\phi)_{\mathsf{T}}$ in Set is as follows:

T	$(\phi)^{\mathsf{T}}$	$(\phi)_{\mathsf{T}}$
Id	ϕ	ϕ
$\underline{K}$	K	$\emptyset$
$\mathsf{T}_1 \times \mathsf{T}_2$	$\{\langle x_1, x_2\rangle \mid x_1 \in (\phi)^{\mathsf{T}_1}\ \wedge\ x_2 \in (\phi)^{\mathsf{T}_2}\}$	$(\{x_1 \mid \exists_{x_2}.\ \langle x_1, x_2\rangle \in \phi\})_{\mathsf{T}_1}$ $\cup\ (\{x_2 \mid \exists_{x_1}.\ \langle x_1, x_2\rangle \in \phi\})_{\mathsf{T}_2}$
$\mathsf{T}_1 + \mathsf{T}_2$	$\{\iota_1\ x_1 \mid x_1 \in (\phi)^{\mathsf{T}_1}\} \cup \{\iota_2\ x_2 \mid x_2 \in (\phi)^{\mathsf{T}_2}\}$	$(\{x_1 \mid \iota_1\ x_1 \in \phi\})_{\mathsf{T}_1}$ $\cup\ (\{x_2 \mid \iota_2\ x_2 \in \phi\})_{\mathsf{T}_2}$
T^K	$\{f \mid \forall_{k \in K}.\ f\ k \in (\phi)^{\mathsf{T}}\}$	$(\{f\ k \mid k \in K\ \wedge\ f \in \phi\})_{\mathsf{T}}$
$\mathcal{P}\mathsf{T}$	$\{t \mid \forall_{x \in \mathsf{T}\ X}\ .\ x \in t\ \Rightarrow x \in (\phi)^{\mathsf{T}}\}$	$(\bigcup \phi)_{\mathsf{T}}$

Jacobs then defines the basic modal operators$\circ$ and $\bullet$ as

$$\begin{aligned} \circ_p\ \phi &= \{u \in U \mid p\ u \in (\phi)^{\mathsf{T}}\} \\ \bullet_p\ \phi &= (\{p\ u \mid u \in \phi\})_{\mathsf{T}} \end{aligned}$$

respectively. The set $\circ_p\ \phi$ contains those states whose immediate successors with respect to p, if any, satisfy ϕ. Similarly $\bullet_p\ \phi$ contains all states which are immediate successors of states satisfying ϕ. This corresponds to the strong *last* operator in modal logic (as the definition of $(\ )_{\mathsf{T}}$ involves an existential quantifier). For each coalgebra p, the operators $\circ_p$ and $\bullet_p$ are related through a Galois connection

$$\bullet_p\ \phi_1 \subseteq \phi_2 \quad \mathsf{iff} \quad \phi_1 \subseteq \circ_p\ \phi_2$$

therefore associating to each p a *Galois algebra* [Kar98], which is essentially a complete Boolean algebra carrying a Galois connection. Therefore, a p-invariant can be defined in two alternative ways as

$$\phi \text{ is a } p\text{-invariant} \quad \mathsf{iff} \quad \phi \subseteq \bigcirc_p \, \phi \quad \mathsf{iff} \quad \bullet_p \, \phi \subseteq \phi$$

The infinite extensions of $\bigcirc_p$ and $\bullet_p$ characterise the (future) 'box' and (past) 'diamond' operators relative to a coalgebra p. Therefore, [Jac99b] introduces $\Box_p \, \phi$ as the *greatest* fixpoint of $\lambda_x \, . \; \phi \cap \bigcirc_p \; x$ and $\blacklozenge_p \; \phi$ as the *least* fixpoint of $\lambda_x \, . \; \phi \cup \bullet_p \; x$. Informally, $\Box_p \; \phi$ reads 'ϕ holds now and in all successor states' whereas $\blacklozenge_p \; \phi$ reads 'ϕ holds now or at some predecessor state'. A simple argument shows that $\Box_p \; \phi$ is the *greatest invariant contained in* ϕ and, dually, $\blacklozenge_p \; \phi$ stands for the *least invariant containing* ϕ. By construction, the two combinators also give rise to a Galois connection.

Although we have already introduced all that is needed in the sequel, this paragraph should not end without underlying the remarkable fact that every coalgebra, for a large class of functors, determine a Galois algebra, thus providing a very precise link between coalgebra theory and (linear) temporal logic. Moreover, a similar result holds for 'branching' functors — *e.g.*, $\mathsf{T} \; X = O \times X^{\mathbf{2}}$. In this case modal operators become indexed by 'navigation' paths in the structure. This link with branching-time temporal logic is explored in [Kur98, Jac99b].

2. Processes as Codata

7. PROCESSES. This section is an attempt to apply the reasoning principles and calculational style underlying the so-called *Bird-Meertens formalism* to the design, in a coalgebraic setting, of process calculi. Processes are taken as inhabitants of the carriers of final coalgebras, whereas process combinators arise as (coinductively defined) morphisms to the carrier of such a coalgebra. This approach has a number of advantages:

- First of all, it provides a *uniform* treatment of processes and other computational structures, *e.g.*, data structures, both represented as categorical types for functors capturing signatures of, respectively, observers and constructors. Placing data and behaviour at a similar level conveys the idea that process models can be chosen and specified according to a given application area, in the same way that a suitable data structure is defined to meet a particular engineering problem. Moreover, processes and data become expressible in programming languages supporting categorical types, such as CHARITY [CF92], providing a convenient way to prototype processes and compare alternative design decisions for their calculi.

- Proofs are carried out in a purely calculational (basically *equational* and *pointfree*) style, therefore circumventing the explicit construction of bisimulations used in most of the literature on process calculi. In particular, a 'conditional fusion' result will be proved in order to handle conditional laws.
- Finally, the approach is independent of any particular process calculus and makes explicit the different ingredients present in the design of such calculi. In particular, structural aspects of the underlying *behaviour model* (*e.g.*, dichotomies such as active *vs* reactive, deterministic *vs* nondeterministic) become clearly separated from the *interaction* structure which defines the synchronisation discipline.

8. Process Structure. In designing a process calculus, its operational semantics is usually given in terms of a transition relation $\xrightarrow{a}$ over processes witnessing the actions in which a process gets committed and the resulting 'continuations', *i.e.*, the behaviours subsequently exhibited. The relation is indexed by the elements a of a set Act of action names, usually equipped with some extra structure to support particular interaction disciplines. For the moment, assume that actions are generated from a set L of *labels*, *i.e.*, a set of formal names. As a rule, the $L \hookrightarrow Act$ embedding $L \hookrightarrow Act$ will be left implicit.

A first design decision concerns the definition of what should be understood by the collection of 'continuation behaviours'. As a rule this is defined as a *set*, in order to express nondeterminism. Other, more restrictive, possibilities consider a sequence or even just a single continuation, modelling, respectively, 'ordered' nondeterminism or determinism. In general, all such different notions will be abstracted into a functor B.

An orthogonal decision concerns the intended interpretation of the transition relation (recall §3.6), usually left implicit or underspecified in process calculi. We may, however, distinguish between

- An 'active' interpretation, in which a transition $p \quad \xrightarrow{a} \quad q$ is informally characterised as 'p evolves to q by performing an action a', both q and a being solely determined by p.
- A 'reactive' interpretation, informally reading 'p reacts to an external stimulus a by evolving to q'.

Processes will then be taken as inhabitants of the carrier of the final coalgebra $\omega : \nu \longrightarrow \mathsf{T}\ \nu$, with T defined as $\mathsf{B}\ (Act \times \mathsf{Id})$, in the 'active' interpretation, and as $(\mathsf{B}\ \mathsf{Id})^{Act}$, in the 'reactive' one.

Following, along this section, the main trend in the process calculi literature, we shall focus on the particular case where B is the finite powerset functor. We shall then consider both

- the 'active' interpretation: processes are elements of the final coalgebra $\omega : \nu \longrightarrow \mathcal{P}(Act \times \nu)$, and the transition relation is defined as

$$p \xrightarrow{a} q \quad \text{iff} \quad \langle a, q\rangle \in \omega\, p$$

- the 'reactive' one, taking processes as elements of the final coalgebra $\omega' : \nu \longrightarrow (\mathcal{P}\nu)^{Act}$ and the transition relation defined as

$$p \xrightarrow{a} q \quad \text{iff} \quad q \in (\omega'\, p)\; a$$

For notation convenience, ω' will be written as $\overline{\omega}$, for $\omega : \nu \times Act \longrightarrow \mathcal{P}\nu$.

The first case, fully developed in the sequel, corresponds to the most common interpretation of, *e.g.*, CCS (as in, for example, [Acz93]). The 'reactive' case will be mentioned only in section 4 together with some other variants to process modelling.

The restriction to the finite powerset avoids cardinality problems and assures the existence of a final coalgebra for T. This means that, we shall be dealing only with *image-finite* processes, a not too severe restriction in practice which may be partially circumvented by a suitable definition of the structure of *Act* (§20).

9. DYNAMIC COMBINATORS. The cornerstone in designing a process calculus is the judicious selection of a (hopefully small) set of process combinators. In [Mil89], R. Milner classifies these into two distinct groups. The first group consists of all combinators which persist through action, *i.e.*, which are present before and after a transition occurs. They are called *static* and used to set up process' architectures, specifying how their components are linked and which parts of their interface are public or private. *Dynamic* combinators, on the other hand, are 'consumed' up on action occurrence, disappearing from the expression representing the process continuation. In the following paragraphs the usual CCS dynamic combinators — *i.e.*, *inaction*, *prefix* and nondeterministic *choice* — are defined as operators on the final universe of processes considered above. Notice that, being non recursive, they have a direct (coinductive) definition which depends solely on the chosen process structure, as detailed in the following paragraphs.

10. INACTION. The inactive process is represented by a constant $\mathsf{nil} : \mathbf{1} \longrightarrow \nu$ upon which no relevant observation can be made. Therefore, it is defined coinductively as

$$\omega \cdot \mathsf{nil} \;=\; \underline{\emptyset}$$

11. PREFIX. The usual way of introducing extension in time in process calculi is by prefixing processes by actions. This is captured by an *Act*-indexed family of operators $a. : \nu \longrightarrow \nu$, for $a \in Act$, defined as

$$\omega \cdot a. \;=\; \mathsf{sing} \cdot \mathsf{label}_a$$

where $\mathsf{label}_a = \langle \underline{a}, \mathsf{id} \rangle$.

12. CHOICE. The actions in the nondeterministic choice of two processes p and q corresponds to the collection of all possible actions of p and q. Therefore, the *choice* operator $+ \; : \; \nu \times \nu \;\longrightarrow\; \nu$ can only be defined in a process model in which the observations form a collection. In particular, *choice* is absent in any deterministic process calculus. For the chosen powerset functor, under the 'active' interpretation, *choice* takes the form

$$\omega \cdot + \;=\; \cup \cdot (\omega \times \omega)$$

13. LEMMA. Structure $\langle \nu; +, \mathsf{nil} \rangle$ is an Abelian idempotent monoid.

Proof. The equalities to be verified are stated below both as (the familiar) equations over process variables p, q and r, and in an equivalent pointfree version which we shall favour in this text:

$$(p+q)+r = p+(q+r) \quad\equiv\quad + \cdot (+ \times \mathsf{id}) = + \cdot (\mathsf{id} \times +) \cdot \mathsf{a} \tag{4.1}$$

$$p + \mathsf{nil} = p \quad\equiv\quad + \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ} = \mathsf{id} \tag{4.2}$$

$$p + p = p \quad\equiv\quad +\cdot\, \Delta = \mathsf{id} \tag{4.3}$$

$$p + q = q + p \quad\equiv\quad + \cdot \mathsf{s} = + \tag{4.4}$$

The verification, in each case, is a simple calculation, resorting to the corresponding properties of set union. Moreover, finality makes ω an isomorphism (§3.39) and therefore, to prove $e = e'$ it is enough to show that $\omega \cdot e = \omega \cdot e'$, because

$$\begin{array}{ll} & \omega \cdot e \;=\; \omega \cdot e' \\ \Rightarrow & \qquad \{ \; \omega \text{ isomorphism} \; \} \\ & \omega^{\circ} \cdot \omega \cdot e \;=\; \omega^{\circ} \cdot \omega \cdot e' \\ \equiv & \qquad \{ \; \omega^{\circ} \cdot \omega = \mathsf{id} \; \} \\ & e \;=\; e' \end{array}$$

—a fact we shall often resort to in the sequel. As an illustration, equation (4.4) is proven as follows:

$$\begin{aligned}
& \omega \cdot + \cdot \mathsf{s} \\
\equiv & \quad \{ \text{ definition } \} \\
& \cup \cdot (\omega \times \omega) \cdot \mathsf{s} \\
\equiv & \quad \{ \text{ s natural } \} \\
& \cup \cdot \mathsf{s} \cdot (\omega \times \omega) \\
\equiv & \quad \{ \cup \text{ commutative } \} \\
& \cup \cdot (\omega \times \omega) \\
\equiv & \quad \{ \text{ definition } \} \\
& \omega \cdot +
\end{aligned}$$

The remaining equalities are proved in [Appendix D, page 321].

□

14. STATIC COMBINATORS. Persistence through action occurrence enforces the recursive definition of *static* combinators. So these will arise as anamorphisms (§3.33) generated by suitable 'gene' coalgebras. Again, we first consider combinators which depend only on the process structure. This is typically the case of *interleaving* and *restriction*.

15. INTERLEAVING. Although *interleaving*, a binary operator $|||\, : \nu \times \nu \longrightarrow \nu$, is not considered as a combinator in most process calculi, it represents the simplest form of 'parallel' aggregation in the sense that it is independent of any particular interaction discipline. The following definition captures the intuition that the observations over the interleaving of two processes correspond to all possible interleavings of the observations of its arguments. Formally,

$$||| \;=\; [\![(\alpha_{|||})]\!]$$

where

$$\begin{aligned}
\alpha_{|||} \;=\; & \nu \times \nu \xrightarrow{\triangle} (\nu \times \nu) \times (\nu \times \nu) \\
& \xrightarrow{(\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)} (\mathcal{P}(Act \times \nu) \times \nu) \times (\nu \times \mathcal{P}(Act \times \nu)) \\
& \xrightarrow{\tau_r \times \tau_l} \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))
\end{aligned}$$

Notice that morphisms $\tau_r : \mathcal{P}(Act \times \nu) \times \nu \longrightarrow \mathcal{P}(Act \times (\nu \times \nu))$ and $\tau_l : \nu \times \mathcal{P}(Act \times \nu) \longrightarrow \mathcal{P}(Act \times (\nu \times \nu))$ used in the last paragraph are, respectively, the right and left strength associated to functor $\mathcal{P}(Act \times \mathsf{Id})$. Therefore, recalling §3.52,

$$\tau_r \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_l \;\text{ and }\; \tau_l \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_r$$

16. LEMMA. The interleaving combinator forms, together with nil, an Abelian monoid.

Proof. As one could expect, proofs of properties of static combinators often resort to the anamorphism fusion law (§3.34). The proof of commutativity is carried out below, as an example. The remaining properties in this lemma are proved in [Appendix D, page 323]. Commutativity states that $p \,|||\, q = q \,|||\, p$, *i.e.*, going pointfree, that $||| \cdot \mathsf{s} = |||$. Now,

$$\begin{array}{cl}
 & ||| \cdot \mathsf{s} \;=\; ||| \\
\equiv & \qquad \{ \text{ definition } \} \\
 & [\!(\alpha_{|||})\!] \cdot \mathsf{s} \;=\; [\!(\alpha_{|||})\!] \\
\Leftarrow & \qquad \{ \text{ ana fusion (3.9) } \} \\
 & \alpha_{|||} \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||}
\end{array}$$

The last equation is justified by the following calculation:

$$\begin{array}{cl}
 & \alpha_{|||} \cdot \mathsf{s} \\
= & \qquad \{ \text{ definition } \} \\
 & \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \vartriangle \cdot \mathsf{s} \\
= & \qquad \{ \vartriangle \text{ natural } \} \\
 & \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot (\mathsf{s} \times \mathsf{s}) \cdot \vartriangle \\
= & \qquad \{ \mathsf{s} \text{ natural } \} \\
 & \cup \cdot (\tau_r \times \tau_l) \cdot (\mathsf{s} \times \mathsf{s}) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \vartriangle \\
= & \qquad \{ \times \text{ functor } \} \\
 & \cup \cdot (\tau_r \cdot \mathsf{s} \times \tau_l \cdot \mathsf{s}) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \vartriangle \\
= & \qquad \{ \tau_r, \tau_l \text{ natural (C.5) and (C.6) } \} \\
 & \cup \cdot (\mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_l \times \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \vartriangle \\
= & \qquad \{ \times \text{ functor } \} \\
 & \cup \cdot (\mathcal{P}(\mathsf{id} \times \mathsf{s}) \times \mathcal{P}(\mathsf{id} \times \mathsf{s})) \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \vartriangle \\
= & \qquad \{ \cup \text{ natural } \}
\end{array}$$

$$\begin{array}{ll}
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \Delta \\
= & \quad \{ \ \cup \text{ commutative } \} \\
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot \mathsf{s} \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \Delta \\
= & \quad \{ \ \mathsf{s} \text{ natural } \} \\
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \mathsf{s} \cdot \Delta \\
= & \quad \{ \text{ routine: } \mathsf{s} \cdot \Delta = \Delta \ \} \\
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \Delta \\
= & \quad \{ \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||}
\end{array}$$

□

17. RESTRICTION. Given a subset $K \subseteq L$, the *restriction* combinator $\backslash_K$ forbids the occurrence of any actions in K. Formally,

$$\backslash_K \ = \ \llbracket\!(\alpha_{\backslash_K})\!\rrbracket$$

where

$$\alpha_{\backslash_K} \ = \ \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathsf{filter}_K} \mathcal{P}(Act \times \nu)$$

where $\mathsf{filter}_K = \lambda s \,.\ \{t \in s |\ \pi_1\ t \notin K\}$.

Restriction commutes with both choice and interleaving. Later we shall see that such results have to be weakened in the presence of interaction. Basic properties of $\backslash_K$ are proved in the next two paragraphs, making use of the following immediate consequence of (3.6): to prove equality $\phi = \psi$, it is enough to show that both $\omega \cdot \phi = \mathsf{T}\ \phi \cdot \alpha$ and $\omega \cdot \psi = \mathsf{T}\ \psi \cdot \alpha$.

18. LEMMA. For any $K \subseteq L$ and $a \in Act$, the interaction of *restriction* with *choice*, *prefix* and *interleaving* is established by the following laws:

$$\backslash_K \cdot + = + \cdot (\backslash_K \times \backslash_K) \qquad (4.5)$$

$$\backslash_K \cdot ||| = ||| \cdot (\backslash_K \times \backslash_K) \qquad (4.6)$$

$$\backslash_K \cdot a. = (a \in K \ \rightarrow \ \mathsf{nil},\ a. \cdot \backslash_K) \qquad (4.7)$$

Proof. [Appendix D, page 325].

□

19. LEMMA. For any $K \subseteq L$,

$$\backslash_K \cdot \backslash_K = \backslash_K \tag{4.8}$$

Proof. [Appendix D, page 328].

□

20. INTERACTION STRUCTURE. Process combinators introduced so far depend solely on the process structure, as recorded in the underlying functor. To specify *interaction*, however, there is a need to introduce some structure on the set *Act* of actions. For this purpose, we axiomatize the *interaction structure* underlying a process calculus as an Abelian positive monoid $\langle Act; \theta, 1\rangle$ with a zero element 0. It is assumed that neither 0 nor 1 belong to the set L of labels. The intuition is that θ determines the *interaction discipline* whereas 0 represents the *absence* of interaction: a zero element is such that, for all $a \in Act$, $a \ \theta \ 0 = 0$. On the other hand, a positive monoid entails $a \ \theta \ a' = 1$ iff $a = a' = 1$.

In some situations 1 may be seen as an *idle* action, but its role, in the general case, is to equip the behaviour functor with a monadic structure, which would not be the case if *Act* were defined simply as an Abelian semigroup. In summary, the role of both 0 and 1 is essentially technical in the description of the interaction discipline. This structure is similar to what is called a *synchronisation algebra* in [WN95] apart from some minor details. In particular, Winskel synchronisation algebras carry a specific constant $\star$ to denote asynchronous occurrence and θ does not necessarily possess a unit. The monoid structure, however, allows for a more uniform characterisation of behaviour models. On the other hand, the definition of parallel composition, in terms of synchronous product and interleaving, avoids the need for $\star$.

21. EXAMPLES. A simple example of an interaction structure captures the notion of action co-occurrence: θ is defined as $a \ \theta \ b = \langle a, b\rangle$, for all $a, b \in Act$ different from 0 and 1. Action equality is defined as that of the 'frontiers' of *Act* terms, in order to assure θ associativity. A formal definition encoded in CHARITY is given in §39 below.

CCS synchronisation discipline [Mil89] provides another example. In this case the set L of labels carries an involutive operation represented by an horizontal bar as in $\overline{a}$, for $a \in L$. Two actions a and $\overline{a}$ are said to be complementary. A special action, denoted by $\tau \notin L$, is introduced to represent the result of a synchronisation between a pair of complementary actions. Therefore, θ evaluates to τ whenever applied to a pair of complementary actions and to 0 in all other cases (except, obviously, if one of the arguments is 1).

Wherever specialising our constructions to the CCS case we follow the standard notational convention under which complements are considered implicitly. In particular, the restriction combinator $\backslash_K$, for $K \subseteq L$ is interpreted as $\backslash_{K \cup \overline{K}}$, where $\overline{K} = \{\overline{a}|\ a \in K\}$. Similarly, the parameter f of a renaming (see below) specifies only the 'action' part, although it is assumed that if $f\ a = b$ then $f\,\overline{a} = \overline{b}$.

As a final remark, note that the structure of *Act* my be further elaborated to increase the expressiveness of the calculus. For example, one may consider actions parametrized by data types entailing an interpretation of the elements of *Act* as *channel* names through which data flow, which corresponds closely to 'CCS with value passing' [Mil89]. Moreover, this allows us to partially circumvent the restriction to image finite processes, mentioned above as a basic limitation of this approach. In fact, only the set of channels, but not that of messages (seen as pairs channel/data) arising in any particular derivation must remain finite.

22. RENAMING. Once an interaction structure is fixed, any homomorphism $f : Act \longrightarrow Act$ lifts to a renaming combinator $[f]$ between processes, defined as

$$[f] = [\![(\alpha_{[f]})]\!]$$

where

$$\alpha_{[f]} = \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathcal{P}(f \times \mathsf{id})} \mathcal{P}(Act \times \nu)$$

23. LEMMA. Renaming preserves identity and composition of homomorphisms, *i.e.*,

$$[\mathsf{id}] = \mathsf{id} \tag{4.9}$$

$$[f] \cdot [f'] = [f \cdot f'] \tag{4.10}$$

for any $f, f' : Act \longrightarrow Act$.

Proof. [Appendix D, page 329].

□

24. LEMMA. Renaming extends along *prefix* and commutes with both *choice* and *interleaving*, *i.e.*,

$$[f] \cdot a. \;=\; (f\ a). \cdot [f] \tag{4.11}$$

$$[f] \cdot + \;=\; + \cdot ([f] \times [f]) \tag{4.12}$$

$$[f] \cdot ||| \;=\; ||| \cdot ([f] \times [f]) \tag{4.13}$$

for any $f : Act \longrightarrow Act, a \in Act$.

Proof. [Appendix D, page 330].

□

25. CONDITIONAL LAWS. Some process equalities hold only if some 'side conditions' are fulfilled. Let us study how such laws are handled in this framework, starting from a very simple example. Let $f = \{b/a\}$ stand for the substitution of a by b, *i.e.*, a homomorphism over Act which is the identity in all actions but a. In several cases, but not in all, renaming with f has no effect. Can this be expressed by a general law? A simple calculation yields

$$\begin{array}{cl}
 & [f] \;=\; \mathsf{id} \\
\equiv & \qquad \{ \text{ renaming definition and ana reflection (3.8) } \} \\
 & \llbracket (\alpha_{[f]}) \rrbracket \;=\; \llbracket (\omega) \rrbracket \\
\Leftarrow & \qquad \{ \text{ ana fusion (3.9) and identity } \} \\
 & \alpha_{[f]} \;=\; \omega
\end{array}$$

Intuitively, the last equality holds for $f = \{b/a\}$ only if a does not show up as an action in the immediate continuations of the process. This condition is formally expressed by the following predicate:

$$\phi \quad = \quad =_{\emptyset} \cdot \cap \cdot \langle \mathcal{P}\pi_1 \cdot \omega, \mathsf{sing} \cdot \underline{a} \rangle$$

Note, however, that ϕ is stated as a local condition on the immediate continuations of any process candidate to satisfy the given equality. Therefore, it cannot be directly taken as a sufficient condition for $[f] = \mathsf{id}$. In fact, to proceed, predicate ϕ has to be made into a β-*invariant* in the sense of §5. This is justified by the following lemma.

26. LEMMA. Let α and β be T-coalgebras and ϕ a predicate on the carrier of β. Then the following 'conditional' fusion law holds

$$(\phi \Rightarrow (\alpha \cdot h = \mathsf{T}\ h \cdot \beta)) \;\;\Rightarrow\;\; (\Box_{\beta}\ \phi \Rightarrow (\llbracket (\alpha) \rrbracket_{\mathsf{T}} \cdot h = \llbracket (\beta) \rrbracket_{\mathsf{T}})) \tag{4.14}$$

Proof. Let X be the carrier of β and i_ϕ the embedding of the subset of X classified by ϕ, *i.e.*, $\phi \cdot i_\phi \;=\; \underline{\mathsf{true}} \cdot !$. Recall that any β-invariant ϕ induces a subcoalgebra β'. Consequently, i_ϕ becomes a comorphism from β' to β. Then

$$\begin{array}{ll}
& \phi \;\Rightarrow\; (\alpha \cdot h = \mathsf{T}\, h \cdot \beta) \\
\equiv & \qquad \{\ i_\phi \text{ definition } \} \\
& \alpha \cdot h \cdot i_\phi \;=\; \mathsf{T}\, h \cdot \beta \cdot i_\phi \\
\Rightarrow & \qquad \{\ \Box_\beta\, \phi \subseteq \phi\ \} \\
& \alpha \cdot h \cdot i_{\Box_\beta\, \phi} \;=\; \mathsf{T}\, h \cdot \beta \cdot i_{\Box_\beta\, \phi} \\
\equiv & \qquad \{\ i_{\Box_\beta\, \phi} \text{ is a comorphism from } \beta' \text{ to } \beta\ \} \\
& \alpha \cdot h \cdot i_{\Box_\beta\, \phi} \;=\; \mathsf{T}\, h \cdot \mathsf{T}\, i_{\Box_\beta\, \phi} \cdot \beta' \\
\equiv & \qquad \{\ \mathsf{T} \text{ functor }\} \\
& \alpha \cdot h \cdot i_{\Box_\beta\, \phi} \;=\; \mathsf{T}\, (h \cdot i_{\Box_\beta\, \phi}) \cdot \beta' \\
\Rightarrow & \qquad \{\ \text{ana fusion (3.9) }\} \\
& [\!(\alpha)\!]_{\mathsf{T}} \cdot h \cdot i_{\Box_\beta\, \phi} \;=\; [\!(\beta')\!]_{\mathsf{T}} \\
\equiv & \qquad \{\ i_{\Box_\beta\, \phi} \text{ being a comorphism implies } [\!(\beta')\!]_{\mathsf{T}} = [\!(\beta)\!]_{\mathsf{T}} \cdot i_{\Box_\beta\, \phi}\ \} \\
& [\!(\alpha)\!]_{\mathsf{T}} \cdot h \cdot i_{\Box_\beta\, \phi} \;=\; [\!(\beta)\!]_{\mathsf{T}} \cdot i_{\Box_\beta\, \phi} \\
\equiv & \qquad \{\ i_\phi \text{ definition }\} \\
& \Box_\beta\, \phi \;\Rightarrow\; ([\!(\alpha)\!]_{\mathsf{T}} \cdot h = [\!(\beta)\!]_{\mathsf{T}})
\end{array}$$

The proof still works if $\Box_\beta\, \phi$ is replaced by any other β-invariant contained in ϕ. As $\Box_\beta\, \phi$ is the greatest such invariant, it provides the most 'generous' condition.

□

27. UNFOLDING $\Box_\beta\, \phi$. Applying the previous lemma to the example of §25, leads to

$$[\{b/a\}] \;=\; \mathsf{id} \;\Leftarrow\; \Box_\omega\, \phi$$

for $\phi = =_\emptyset \cdot \cap \cdot \langle \mathcal{P}\pi_1 \cdot \omega, \mathsf{sing} \cdot \underline{a} \rangle$. Recall from §5 that $\Box_\omega\, \phi$ is defined as the greatest fixpoint of

$$\Phi \;=\; \lambda x \,.\; \phi \cap \bigcirc_\omega\, x$$

Looking at predicates as sets, Φ is a function over a complete lattice — $\langle \mathcal{P}\nu, \subseteq \rangle$ — whose monotony is easily proved by induction on the functor structure. Therefore, a concrete representation for $\Box_\omega\, \phi$ can be computed by the Knaster-Tarski theorem

[Tar55] as the union of all post-fixpoints of Φ, *i.e.*,

$$\Box_\omega\ \phi\ =\ \bigcup\{s \in \mathcal{P}\nu|\ s \subseteq \phi \cap \bigcirc_\omega\ s\}$$

Being a post-fixpoint means that, for each s above and any process p,

$$\begin{aligned}
& p \in s\ \Rightarrow\ p \in \phi\ \wedge\ p \in \bigcirc_\omega\ s \\
\equiv\ & p \in \phi\ \wedge\ p \in \{x \in \nu|\ \omega\ x \in (s)^{\mathcal{P}(Act \times \_)}\} \\
\equiv\ & p \in \phi\ \wedge\ p \in \{x \in \nu|\ \omega\ x \in \{c \in \mathcal{P}(Act \times \nu)|\ \forall_{t \in Act \times \nu}\ .\ t \in c \Rightarrow t \in (s)^{Act \times \_}\}\} \\
\equiv\ & p \in \phi\ \wedge\ p \in \{x \in \nu|\ \omega\ x \in \{c \in \mathcal{P}(Act \times \nu)|\ \forall_{t \in Act \times \nu}\ .\ t \in c \Rightarrow \pi_2\ t \in s\}\} \\
\equiv\ & p \in \phi\ \wedge\ p \in \{x \in \nu|\ (\mathcal{P}\pi_2 \cdot \omega)\ x \in s\}
\end{aligned}$$

Seen as a set, predicate ϕ is given by $\phi = \{x \in \nu|\ (\mathcal{P}\pi_1 \cdot \omega)\ x \cap \{a\} = \emptyset\}$. Therefore,

$$\Box_\omega\ \phi\ =\ \bigcup\{s \in \mathcal{P}\nu|\ \forall_{x \in \nu}\ .\ x \in s \Rightarrow ((\mathcal{P}\pi_1 \cdot \omega)\ x \cap \{a\} = \emptyset) \wedge ((\mathcal{P}\pi_2 \cdot \omega)\ x \in s))\}$$

In words, this is the set of all processes whose derivations never exhibit action a. Also notice our attention can be restricted to actions embedded from L. In fact, as f is, by definition, a homomorphism in *Act*, it is necessarily the identity on 0, 1 or any other 'special' action introduced as a constant in *Act*.

In CCS, the set of all labels (seen as actions) in which a process p can commit itself, *i.e.*, that appear in at least one derivation of p, is called the *sort* of p and denoted by $\mathcal{L}(p)$. [Mil89] provides a syntactic criterion to compute a majoring approximation of $\mathcal{L}(p)$ by induction on the process expression. A semantic definition can, however, be given as

$$\mathcal{L}(p) = (\mathsf{filter}_{Act-L} \cdot \mathcal{P}\pi_1 \cdot \bigcup \cdot \mathcal{P}p)\ \Box_p\ \mathsf{true}$$

where filter is as defined in §17 and, again, the embedding of L in *Act* is left implicit. The law under study may then be rewritten as

$$[\{b/a\}]\ =\ \mathsf{id}\ \Leftarrow\ \notin_a \cdot \mathcal{L}$$

where $\notin_a$ is defined as $\lambda x\ .\ a \notin x$. Going pointwise we end up with the familiar CCS law $p\ [\{b/a\}]\ =\ p\ \ \Leftarrow\ a, \overline{a} \notin \mathcal{L}(p)$ (by convention, the parameter of the CCS renaming operator represents 'coactions' implicitly, *cf.* §21).

28. PRODUCT. This paragraph introduces another static process combinator, *synchronous product*, which corresponds to the simultaneous execution of two processes. In each step, the resulting action is determined by the interaction structure for the calculus. Formally,

$$\otimes\ =\ [\![\alpha_\otimes]\!]$$

where

$$\alpha_{\otimes} \;=\; \nu \times \nu \xrightarrow{(\omega \times \omega)} \mathcal{P}(Act \times \nu) \times \mathcal{P}(Act \times \nu) \xrightarrow{\delta_r}$$
$$\mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\mathsf{sel}} \mathcal{P}(Act \times (\nu \times \nu))$$

This definition involves the distribution law for strong monads (§A.10) and $\mathsf{sel} = \mathsf{filter}_{\{0\}}$, which filters out all synchronisation failures (recall 0 is the zero element of the interaction structure, representing absence of synchronisation).

29. REMARK. In the definition above, interaction is catered by δ_r — the distributive law for the strong monad $\mathcal{P}(Act \times \mathsf{Id})$ (see §A.10). In fact, the monoidal structure in Act adopted in §20 extends functor $\mathcal{P}(Act \times \mathsf{Id})$ to a strong monad. Moreover, as θ is commutative, so is the resulting monad. This makes it irrelevant to choose δ_r or its left version δ_l in the definition of $\otimes$. Recall, from appendix A, that δ_r amounts to the composition of the left and the right strengths in the correspondent Kleisli category. On its turn, Kleisli composition (§A.7) involves the application of the monad multiplication μ to 'flatten' the result. For a monoid monad, this requires the suitable application of the underlying monoidal operation, which, in our case, fixes the interaction discipline. To make the point clear, let us explicitly describe δ_r for this monad.

$$\begin{aligned}
& \delta_r^{\mathcal{P}(Act \times \mathsf{Id})} \\
= \quad & \{\ \delta_r \text{ definition}\ \} \\
& \tau_r^{\mathcal{P}(Act \times \mathsf{Id})} \bullet \tau_l^{\mathcal{P}(Act \times \mathsf{Id})} \\
= \quad & \{\ \bullet \text{ definition}\ \} \\
& \mu^{\mathcal{P}(Act \times \mathsf{Id})} \cdot \mathcal{P}(\mathsf{id} \times \tau_r^{\mathcal{P}(Act \times \mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act \times \mathsf{Id})} \\
= \quad & \{\ \mu \text{ for } \mathcal{P}(Act \times \mathsf{Id})\ \} \\
& \mathcal{P}\mu^{(Act \times \mathsf{Id})} \cdot \mu^{\mathcal{P}} \cdot \mathcal{P}\tau_l^{\mathcal{P}} \cdot \mathcal{P}(\mathsf{id} \times \tau_r^{\mathcal{P}(Act \times \mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act \times \mathsf{Id})} \\
= \quad & \{\ \mu \text{ for } (Act \times \mathsf{Id})\ \} \\
& \mathcal{P}((\theta \times \mathsf{id}) \cdot \mathsf{a}^{\circ}) \cdot \mu^{\mathcal{P}} \cdot \mathcal{P}\tau_l^{\mathcal{P}} \cdot \mathcal{P}(Act \times \tau_r^{\mathcal{P}(Act \times \mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act \times \mathsf{Id})} \\
= \quad & \{\ \mu^{\mathcal{P}} \text{ natural and definition}\ \} \\
& \bigcup \cdot \mathcal{P}(\mathcal{P}((\theta \times \mathsf{id}) \cdot \mathsf{a}^{\circ}) \cdot \tau_l^{\mathcal{P}}) \cdot \mathcal{P}(Act \times \tau_r^{\mathcal{P}(Act \times \mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act \times \mathsf{Id})}
\end{aligned}$$

i.e., going pointwise,

$$\delta_r^{\mathcal{P}(Act \times \mathsf{Id})}\ \langle c_1, c_2 \rangle \;=\; \{\langle a'\,\theta\,a, \langle p, p' \rangle\rangle \mid \langle a, p \rangle \in c_1 \,\wedge\, \langle a', p' \rangle \in c_2\}$$

30. LEMMA. Synchronous product is commutative, associative and has nil as a zero element, *i.e.*,

$$\otimes \cdot \mathsf{s} \;=\; \otimes \tag{4.15}$$

$$\otimes \cdot (\otimes \times \mathsf{id}) \;=\; \otimes \cdot (\mathsf{id} \times \otimes) \cdot \mathsf{a} \tag{4.16}$$

$$\otimes \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ} \;=\; \mathsf{nil} {\cdot} ! \tag{4.17}$$

Proof. [Appendix D, page 333].

□

31. LEMMA. Synchronous product distributes over *choice* and, conditionally, over *restriction* and *renaming*, *i.e.*, for any $K \subseteq L$ and renaming homomorphism f,

$$\otimes \cdot (\mathsf{id} \times +) = + \cdot (\otimes \times \otimes) \cdot \mathsf{pdr} \tag{4.18}$$

$$\backslash_K \cdot \otimes = \otimes \cdot (\backslash_K \times \backslash_K) \qquad \Leftarrow \; \mathsf{uniform\_restriction} \tag{4.19}$$

$$[f] \cdot \otimes = \otimes \cdot ([f] \times [f]) \qquad \Leftarrow \; \mathsf{non\_collapse\_renaming} \tag{4.20}$$

where $\mathsf{pdr} : X \times (Y \times Z) \longrightarrow (X \times Y) \times (X \times Z)$ abbreviates $\langle \mathsf{id} \times \pi_1, \mathsf{id} \times \pi_2 \rangle$. The first law, which going pointwise reads

$$p \otimes (q + r) \;=\; (p \otimes q) + (p \otimes r)$$

is characteristic of the synchronous product but (obviously) fails for *interleaving*. So it does not generalise to the usual parallel composition found in process calculi.

Proof. We shall prove (4.19) below, as an example of how the side condition is derived from the proof of a conditional law. The remaining cases are proved in [Appendix D, page 336]. We begin by unfolding expression $\omega \cdot \otimes \cdot (\backslash_K \times \backslash_K)$:

$$\begin{array}{cl}
 & \omega \cdot \otimes \cdot (\backslash_K \times \backslash_K) \\
= & \qquad \{ \text{ comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_{\otimes} \cdot (\backslash_K \times \backslash_K) \\
= & \qquad \{ \otimes \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot (\backslash_K \times \backslash_K) \\
= & \qquad \{ \times \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \cdot \backslash_K \times \omega \cdot \backslash_K) \\
= & \qquad \{ \text{ comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K}) \times (\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K})
\end{array}$$

$$\begin{aligned}
= &\qquad \{ \times \text{ functor} \} \\
&\mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(\mathsf{id} \times \backslash_K) \times \mathcal{P}(\mathsf{id} \times \backslash_K)) \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= &\qquad \{ \delta_r \text{ natural} \} \\
&\mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= &\qquad \{ \mathsf{sel} = \mathsf{filter}_{\{0\}} \text{ and } \mathsf{filter} \text{ natural} \} \\
&\mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= &\qquad \{ \times \text{ functor} \} \\
&\mathcal{P}(\mathsf{id} \times \otimes \cdot (\backslash_K \times \backslash_K)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})
\end{aligned}$$

Next a similar calculation is done for $\omega \cdot \backslash_K \cdot \otimes$, trying to arrive to an expression $\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \alpha$, with $\alpha = \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})$ as above.

$$\begin{aligned}
&\omega \cdot \backslash_K \cdot \otimes \\
= &\qquad \{ \text{comorphism} \} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K} \cdot \otimes \\
= &\qquad \{ \alpha_{\backslash_K} \text{ definition} \} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \cdot \otimes \\
= &\qquad \{ \text{comorphism} \} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_\otimes \\
= &\qquad \{ \alpha_\otimes \text{ definition} \} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \\
= &\qquad \{ \mathsf{filter}_K \text{ natural} \} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{filter}_K \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \\
\overset{?}{=} &\qquad \{ \star \} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \times \mathsf{filter}_K) \cdot (\omega \times \omega) \\
= &\qquad \{ \times \text{ functor} \} \\
&\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \cdot \omega \times \mathsf{filter}_K \cdot \omega) \\
= &\qquad \{ \alpha_{\backslash_K} \text{ definition} \} \\
&\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})
\end{aligned}$$

We have succeeded only partially: the step marked with a $\star$ does not hold universally. We are left with the task of establishing under what conditions, if any, the following equality holds:

$$\mathsf{filter}_K \cdot \mathsf{sel} \cdot \delta_r \;=\; \mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \times \mathsf{filter}_K)$$

Unfolding definitions and going pointwise for a while, we get

$$\begin{aligned}
&(\,\mathsf{filter}_K \cdot \mathsf{sel} \cdot \delta_r)\,\langle c_1, c_2\rangle \;=\\
&\quad= (\mathsf{filter}_K \cdot \mathsf{sel})\,\{\langle a'\,\theta\,a, \langle p,p'\rangle\rangle \in Act \times (\nu \times \nu) |\ \langle a,p\rangle \in c_1 \ \wedge\ \langle a',p'\rangle \in c_2\}\\
&\quad= \mathsf{filter}_K\,\{\langle a'\,\theta\,a, \langle p,p'\rangle\rangle \in Act \times (\nu \times \nu) |\ \langle a,p\rangle \in c_1 \ \wedge\ \langle a',p'\rangle \in c_2 \ \wedge\ a'\,\theta\,a \neq 0\}\\
&\quad= \{\langle a'\,\theta\,a, \langle p,p'\rangle\rangle |\ \langle a,p\rangle \in c_1 \ \wedge\ \langle a',p'\rangle \in c_2 \ \wedge\ a'\,\theta\,a \neq 0 \ \wedge\ a'\,\theta\,a \notin K\}
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&(\,\mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \times \mathsf{filter}_K))\,\langle c_1, c_2\rangle \;=\\
&\quad= (\mathsf{sel} \cdot \delta_r)\ \langle\{\langle a,p\rangle \in c_1 |\ a \notin K\}, \{\langle a',p'\rangle \in c_2 |\ a' \notin K\}\rangle\\
&\quad= \mathsf{sel}\ \{\langle a'\,\theta\,a, \langle p,p'\rangle\rangle \in Act \times (\nu \times \nu) |\ \langle a,p\rangle \in c_1 \ \wedge\ \langle a',p'\rangle \in c_2 \ \wedge\ a \notin K \ \wedge\ a' \notin K\}\\
&\quad= \{\langle a'\,\theta\,a, \langle p,p'\rangle\rangle |\ \langle a,p\rangle \in c_1 \ \wedge\ \langle a',p'\rangle \in c_2 \ \wedge\ a \notin K \ \wedge\ a' \notin K \ \wedge\ a'\,\theta\,a \neq 0\}
\end{aligned}$$

Clearly the two sets can be identified iff, for all possible a and a', such that $a'\ \theta\ a \neq 0$, $a'\,\theta\,a \notin K \equiv a \notin K \wedge a' \notin K$. Therefore, step $\star$ is only possible if the expression scope is restricted to pairs of processes satisfying the following predicate:

$$\phi\ \langle p,q\rangle \;=\; \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p,\, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a\,\theta\,a' \neq 0 \;\Rightarrow\; (a\,\theta\,a' \notin K \;\equiv\; a \notin K \wedge a' \notin K)$$

which is lifted to the invariant

$$\mathsf{uniform\_restriction} \;=\; \Box_\alpha\ \phi$$

□

In §36 we investigate the implication of this invariant for particular interaction structures such as those of Ccs and Csp.

32. Parallel. Let us finally address the *parallel composition* process combinator. This is defined operationally in Ccs by the following inference rules:

$$\frac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q} \qquad\qquad \frac{q \xrightarrow{a} q'}{p \mid q \xrightarrow{a} p \mid q'} \qquad\qquad \frac{p \xrightarrow{a} p' \quad q \xrightarrow{\overline{a}} q'}{p \mid q \xrightarrow{\tau} p' \mid q'}$$

This definition conveys the intuition that the evolution of $p \mid q$ consists of all possible derivations of p and q plus the ones associated to the synchronisations allowed by the particular interaction structure for Ccs, as described in §21. In general, parallel composition can be expressed in terms of *interleaving* and *synchronous product*, the last being, as discussed above, suitably parametrized by the *interaction structure*. The required combination of $|||$ and $\otimes$ is 'genetic' in the sense that is performed at the level of the 'genes' for $|||$ and $\otimes$, as follows:

$$| \;=\; [\![(\alpha_{|})]\!]$$

where

$$\alpha_{|} \;=\; \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\alpha_{|||} \times \alpha_{\otimes})}$$
$$\mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))$$

33. PARALLEL LAWS. As expected, parallel composition shares some properties with both $|||$ and $\otimes$. In particular, it gives rise, with nil, to an Abelian monoid and distributes along renaming and restriction in certain cases. On the other hand, it lacks a zero element and does not distribute through choice. We focus our attention below to the proof of two of these properties: commutativity and (conditional) distribution through restriction. In both cases, the 're-use' of the proofs of the corresponding results for $|||$ and $\otimes$ will be emphasised.

34. LEMMA. Parallel composition is commutative, *i.e.*, $|\cdot \mathsf{s} = |$.

Proof.

$$
\begin{array}{ll}
 & | \cdot \mathsf{s} \;= | \\
\equiv & \qquad \{ \; | \text{ definition } \} \\
 & [\![(\alpha_{|})]\!] \cdot \mathsf{s} \;=\; [\![(\alpha_{|})]\!] \\
\Leftarrow & \qquad \{ \text{ ana fusion (3.9) } \} \\
 & \alpha_{|} \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|}
\end{array}
$$

and the last equation holds because

$$
\begin{array}{ll}
 & \alpha_{|} \cdot \mathsf{s} \\
= & \qquad \{ \; \alpha_{|} \text{ definition } \} \\
 & \cup \cdot (\alpha_{|||} \times \alpha_{\otimes}) \cdot \; \Delta \; \cdot \mathsf{s} \\
= & \qquad \{ \; \Delta \text{ natural } \} \\
 & \cup \cdot (\alpha_{|||} \times \alpha_{\otimes}) \cdot (\mathsf{s} \times \mathsf{s}) \cdot \; \Delta \\
= & \qquad \{ \text{ reusing } \alpha_{|||} \cdot \mathsf{s} = \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||} \text{ and } \alpha_{\otimes} \cdot \mathsf{s} = \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{\otimes}
\end{array}
$$

from proofs of §16 and (4.15) in §30 }

$$\begin{aligned}
& \cup \cdot (\mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||} \times \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{\otimes}) \cdot \triangle \\
= \quad & \{ \times \text{ functor and } \cup \text{ natural} \} \\
& \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\alpha_{|||} \times \alpha_{\otimes}) \cdot \triangle \\
= \quad & \{ \alpha_{|} \text{ definition} \} \\
& \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|}
\end{aligned}$$

□

35. LEMMA. Parallel composition interacts with restriction according to

$$\backslash_K \cdot | \;=\; | \cdot (\backslash_K \times \backslash_K) \qquad \Leftarrow \mathsf{uniform\_restriction}'$$

for any $K \subseteq L$.

Proof. As usual, we try to equate the composites of ω with both sides of the equation to identify a common coalgebra α'. Notice that a similar argument has been used in previous paragraphs to show a similar result about ||| and ⊗. Thus,

$$\begin{aligned}
& \omega \cdot | \cdot (\backslash_K \times \backslash_K) \\
= \quad & \{ \text{comorphism} \} \\
& \mathcal{P}(\mathsf{id} \times |) \cdot \alpha_{|} \cdot (\backslash_K \times \backslash_K) \\
= \quad & \{ \alpha_{|} \text{ definition} \} \\
& \mathcal{P}(\mathsf{id} \times |) \cdot \cup \cdot (\alpha_{|||} \times \alpha_{\otimes}) \cdot \triangle \cdot (\backslash_K \times \backslash_K) \\
= \quad & \{ \text{reusing proofs of (4.6) (§18) and (4.19) (§31)} \} \\
& \mathcal{P}(\mathsf{id} \times |) \cdot \cup \cdot \\
& \quad ((\mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K})) \cdot \triangle) \\
& \quad\; \times \\
& \quad (\mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \cdot \alpha_{\backslash_K})) \\
& \cdot \triangle \\
= \quad & \{ \cup \text{ naturality} \} \\
& \mathcal{P}(\mathsf{id} \times |) \cdot \mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \cdot \cup \\
& \quad ((\cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K})) \cdot \triangle) \\
& \quad\; \times \\
& \quad \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \cdot \alpha_{\backslash_K}))
\end{aligned}$$

$$\cdot\ \Delta$$

$$= \qquad \{\ \text{functors}\ \}$$

$$\mathcal{P}(\mathsf{id} \times | \cdot (\backslash_K \times \backslash_K)) \cdot \cup$$

$$((\cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K}))\cdot\ \Delta)$$

$$\times$$

$$\mathsf{sel}\ \cdot \delta_r \cdot (\alpha_{\backslash_K} \cdot \alpha_{\backslash_K}))$$

$$\cdot\ \Delta$$

On the other hand,

$$\omega \cdot \backslash_K \cdot |$$

$$= \qquad \{\ \text{comorphism}\ \}$$

$$\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K} \cdot |$$

$$= \qquad \{\ \alpha_{\backslash_K}\ \text{definition}\ \}$$

$$\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \cdot |$$

$$= \qquad \{\ \text{comorphism}\ \}$$

$$\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times |) \cdot \alpha_|$$

$$= \qquad \{\ \alpha_|\ \text{definition}\ \}$$

$$\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times |) \cdot \cup \cdot (\alpha_{|||} \times \alpha_{\otimes})\cdot\ \Delta$$

$$= \qquad \{\ \mathsf{filter}_K\ \text{natural}\ \}$$

$$\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times |) \cdot \mathsf{filter}_K \cdot \cup \cdot (\alpha_{|||} \times \alpha_{\otimes})\cdot\ \Delta$$

$$= \qquad \{\ \cup\ \text{natural and}\ \mathcal{P}\ \text{functor}\ \}$$

$$\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |)) \cdot \cup \cdot (\mathsf{filter}_K \times \mathsf{filter}_K) \cdot (\alpha_{|||} \times \alpha_{\otimes})\cdot\ \Delta$$

$$= \qquad \{\ \times\ \text{functor}\ \}$$

$$\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |)) \cdot \cup \cdot (\mathsf{filter}_K \cdot \alpha_{|||} \times \mathsf{filter}_K \cdot \alpha_{\otimes})\cdot\ \Delta$$

$$= \qquad \{\ \text{reusing proofs of (4.6) (§18) and (4.19) (§31)}\ \}$$

$$\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |) \cdot \cup \cdot$$

$$((\cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K}))\cdot\ \Delta)$$

$$\times$$

$$\mathsf{sel}\ \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}))$$

$$\cdot\ \Delta$$

At this point a common coalgebra has been identified:

$$\alpha' \ = \ \cup \cdot ((\cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K}))\cdot\ \Delta) \times \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}))\cdot\ \Delta$$

and, thus, we are almost ready to conclude. However, it has to be remarked that, on reusing the proof of (4.19) in the last step of this derivation, we must also take into account the predicate which constrains the substitution made. Clearly, the same predicate ϕ derived in §31 is again the local condition requested to validate the derivation here. We may then conclude the validity of the law, subject to the restriction given by

$$\mathsf{uniform\_restriction}' \;=\; \Box_{\alpha'}\;\phi$$

□

36. We may now ask what form invariants $\mathsf{uniform\_restriction}$ and $\mathsf{uniform\_restriction}'$ take given a particular interaction structure. For example θ has only three possible results under CCS's interaction discipline — τ, 1 and 0 — and so the result of θ never belongs to L. Therefore, as $K \subseteq L$, condition $a'\theta a \notin K$ holds and ϕ becomes

$$
\begin{array}{ll}
& \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p,\, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a\,\theta\,a' \neq 0 \;\Rightarrow\; a\,\theta\,a' \notin K \;\equiv\; a \notin K \wedge a' \notin K \\
\equiv & \qquad \{\ \text{CCS interaction structure}\ \} \\
& \forall_{a \in \mathcal{P}(\pi_1 \cdot \omega)\ p,\, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ (a\,\theta\,a' = \tau \vee a\,\theta\,a' = 1) \;\Rightarrow\; a \notin K \wedge a' \notin K \\
\equiv & \qquad \{\ a\,\theta\,a' = 1\ \text{ iff }\ a = a' = 1\ \} \\
& \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p,\, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a\,\theta\,a' = \tau \;\Rightarrow\; a \notin K \wedge a' \notin K \\
\equiv & \qquad \{\ a\,\theta\,a' = \tau\ \text{ iff }\ a' = \overline{a}\ \} \\
& \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p,\, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a' = \overline{a} \;\Rightarrow\; a \notin K \wedge \overline{a} \notin K \\
\equiv & \qquad \{\ \text{rearranging}\ \} \\
& (\mathcal{P}\pi_1 \cdot \omega)\ p \cap \overline{(\mathcal{P}\pi_1 \cdot \omega)\ q} \cap (K \cup \overline{K}) = \emptyset
\end{array}
$$

where the overbar notation stands here for the lifting of the involutive CCS complement operation to sets of actions.

Another question concerns the relation between $\mathsf{uniform\_restriction}'$ and the predicate $\mathsf{uniform\_restriction}$ arising in the proof of a similar law for $\otimes$ in §31. Although both of them are defined after the same local condition ϕ, they stand for closures under different coalgebras and are, consequently, distinct predicates. Formally,

$$
\begin{array}{l}
\mathsf{uniform\_restriction} \;=\; \Box_{\alpha}\;\phi \qquad \text{for } \alpha = \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \cdot \alpha_{\backslash_K}) \\
\mathsf{uniform\_restriction}' \;=\; \Box_{\alpha'}\;\phi \qquad \text{for } \alpha' = \\
\quad \cup \cdot ((\cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K})) \cdot \triangle) \times \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})) \cdot \triangle
\end{array}
$$

In fact, the number of derivations that have to be considered under ϕ is much greater in the second case. In particular, $\mathsf{uniform\_restriction}$ does not consider configurations representing interleavings. For example, if one of the processes exhausts after n steps, the condition on the actions is not required to hold after that. On the other hand, in $\mathsf{uniform\_restriction}'$, ϕ is closed wrt the transitions on a', which include both interleavings and synchronisations. Therefore, and recalling the notion of *sort*, we conclude that $\mathsf{uniform\_restriction}' \;\equiv\; \mathcal{L}(p) \cap \overline{\mathcal{L}(p)} \cap (K \cup \overline{K}) \;=\; \emptyset$ arriving to the familiar CCS presentation of this result:

$$(p \mid q)\backslash_K = p\,\backslash_K \,|q\,\backslash_K \qquad \Leftarrow \mathcal{L}(p) \cap \overline{\mathcal{L}(p)} \cap (K \cup \overline{K}) = \emptyset$$

This reasoning illustrates the claim that our approach to process calculi allows us to 'discover' the appropriate restriction a law is constrained by, instead of 'postulating' it and verifying its suitability. It also makes explicit that such conditions are essentially dependent only on the calculus underlying interaction structure. Consider, for example, what would happen to the law at hands should a CSP-like interaction discipline be adopted instead. In CSP [Hoa85] only equally named actions synchronise, leading to the following definition of θ:

$$a\,\theta\,a \;=\; a\,, \quad a\,\theta\,1 \;=\; 1\,\theta\,a \;=\; a \quad \text{and} \quad a\,\theta\,b \;=\; 0 \quad \text{in all other cases}$$

Therefore, condition $a\;\theta\;a' \neq 0 \Rightarrow (a\;\theta\;a' \notin K \;\;\equiv\;\; a \notin K \,\wedge\, a' \notin K)$ becomes trivially true and the law holds without any side condition.

37. ANOTHER EXAMPLE. Another example involving the derivation of side conditions appeared in the proof of (4.20) in §31. The situation is similar to the one discussed above as a similar result can be stated for | and the required condition results again from the same local condition

$$\phi\;\langle p, q\rangle \;=\; \forall_{a \in (\mathcal{P}p1\cdot\omega)\;p,\, a' \in (\mathcal{P}p1\cdot\omega)\;q}\;.\; a\,\theta\,a' \neq 0 \;\equiv\; fa\;\theta\,fa' \neq 0$$

Let us briefly examine its form under the CCS interaction discipline. In such a setting, $a\,\theta\,a' \neq 0$ implies that $a\,\theta\,a' = 1$ or $a\,\theta\,a' = \tau$. Hence, when does the equivalence $a\,\theta\,a' \neq 0 \;\equiv\; fa\;\theta\;fa' \neq 0$ hold? Clearly, the implication from left to right holds trivially because f is a homomorphism on the interaction structure:

$$\begin{array}{ll}
 & a\,\theta\,a' \;=\; \star \\
\Rightarrow & \qquad \{\ \text{Leibniz } (\star \text{ stands for either } 1 \text{ or } \tau)\ \} \\
 & f\;(a\,\theta\,a') \;=\; f\;\star \\
\equiv & \qquad \{\ f \text{ is an } \mathit{Act}\text{-homomorphism}\ \} \\
 & fa\;\theta\,fa' \;=\; \star
\end{array}$$

The implication in the opposite direction, however, reads

$$fa\ \theta\ fa' = \star\ \Rightarrow\ a\,\theta\,a'\ =\ \star$$

which, f being an *Act*-homomorphism, is equivalent to

$$f\ (a\,\theta\,a')\ =\ f\ \star\ \Rightarrow a\,\theta\,a'\ =\ \star$$

again for $\star$ standing for either 1 or τ. This holds only if f is *mono*. Thus, for the CCS case, the generated invariant — non_collapse_renaming — requires the injectivity of f or, at least, of its restriction to the relevant process sorts. The resulting law is usually written in CCS as follows:

$$(p \mid q)[f] = p[f] \mid q[f] \qquad \Leftarrow\ f \text{ restricted to } \mathcal{L}(p \mid q) \cup \overline{\mathcal{L}(p \mid q)} \text{ is mono}$$

3. Prototyping Processes

38. PROCESS STRUCTURE. One advantage of thinking about processes as inhabitants of (coinductive) types is the possibility of developing prototypes for process calculi in functional languages supporting such types. Once a prototype implementation of a particular calculus is developed, processes can be defined in the language and their execution traced. Of course prototyping is not a formal proof procedure. However, experiments can be carried out within the calculus which allow the experimenter to observe processes' evolution and eventually to assess different design decisions.

In this section the process combinators defined earlier in this chapter are presented as CHARITY programs. The reader is referred to appendix E for an introduction to the language. The direct correspondence between the formal definitions and the CHARITY code makes detailed explanations of the prototypes unnecessary. Just notice that some 'housekeeping' morphisms, like a or l used earlier on, are more conveniently handled by the CHARITY term logic.

Our starting point is the following declaration of a process space as the coinductive type Pr(A), parametrized by a specification A of the interaction structure. In CHARITY,

```
data C -> Pr(A) = bh: C -> set(A * C).
```

The following paragraphs provide two possible specifications of the interaction discipline followed by an encoding of the process combinators introduced earlier on. Finally, from §44 onwards, we introduce a term language for processes and show how an interpreter for it can be easily derived by resorting, again, to universal constructions.

39. INTERACTION STRUCTURES. As discussed in §8 and 20, *interaction structures* are parametrized by a set L of labels. Therefore, for each process calculus, actions over L are introduced as an inductive type `Ac(L)` upon which an equality function and a product θ are defined. Two interaction structures are defined below, corresponding to the examples in §21. In the first case the product of two actions `a` and `b`, different from 0 and 1, corresponds to their co-occurrence and is represented by `syn(a,b)`. The second case, on the other hand, captures CCS interaction discipline.

CO-OCCURRENCE. In this case no special structure is assumed on the set of labels `L`. The type of actions, parametric on `L`, is therefore defined inductively as follows

```
data Ac(L) -> A =
 act: L -> A | syn: A * A -> A | nop: 1 -> A | idle: 1 -> A.
```

The embedding of labels into actions is explicitly represented by constructor `act`. The distinguished actions 0 and 1 are denoted by `nop` and `idle`, respectively. The specification is complete with a definition of action product θ, encoded below as function `prodAc`, and an equality predicate `eqA` on actions, both parametric on `L`. The actual CHARITY code for θ is as follows,

```
def prodAc : Ac(L) * Ac(L) -> Ac(L)
    = (nop, _ ) => nop  | (_ ,nop) => nop | (idle, x) => x
    | (x ,idle) => x    | (a1, a2) => syn(a1, a2).
```

Comparing two actions amounts to compute the ‘contour’ of the trees representing the corresponding terms, as mentioned in §21. Such a function is, of course, a `Ac(L)` catamorphism. Thus,

```
def eqA{eqL: L * L -> bool} : Act(L) * Act(L) -> bool
    =  (a1, a2) =>  eq_list{eqL} (contour a1, contour a2).


def contour: Act(L) ->  list(L)
     = a =>  {| nop: ()   => [nop] | idle: ()  => [idle]
              | act: l    => [l]
              | syn: (x,y) => append (x,y)
              |} a.
```

CCS. The specification of an interaction structure for CCS requires the introduction of the involutive complement operation on labels. Thus,

```
data Lb(N) -> I = name: N -> I | inv: I -> I.

def eqL{eqN: N * N -> bool} : Lb(N) *  Lb(N) -> bool
    = (name(m), name(n))           => eqN(m,n)
    | (inv(name(m)), inv(name(n))) => eqN(m,n)
    | _                            => false.
```

Then the algebra of actions is defined as

```
data Ac(L) -> A =
 act: L -> A | tau: 1 -> A | nop: 1 -> A | idle: 1 -> A.
```

together with

```
def eqA{eqL: L * L -> bool} : Ac(L) * Ac(L) ->bool
    = (act(l), act(k))     => eqL(l,k)
    | (tau, tau)           => true
    | (nop, nop)           => true
    | (idle, idle)         => true
    | _                    => false.


def prodAc{eqL: Lb(N) * Lb(N) -> bool} :
          Ac(Lb(N)) * Ac(Lb(N)) ->  Ac(Lb(N))
    = (act(l), act(k)) => { true => tau | false => nop }
                          or(eqL(l,inv(k)), eqL(inv(l),k))
    | (idle, x)             => x
    | (x, idle)             => x
    | _                     => nop.
```

40. Dynamic Combinators. The next step is the introduction of the three dynamic combinators, *inaction*, *prefix* and *choice*. Their formal definitions (in §10, §11 and §12) are directly translated to Charity, originating functions `bnil`, `bpre` and `bcho`, respectively.

```
def bnil: 1 -> Pr(A)
    = () => (bh: empty).

def bpre: A * Pr(A) -> Pr(A)
    = (a, t)  => (bh: sing(a,t)).

def bcho: Pr(A) * Pr(A)  -> Pr(A)
    = (t1, t2) => (bh: union(bh t1, bh t2)).
```

41. STATIC COMBINATORS. Also in this case the formal definitions are directly translated to the prototyping language. Compare, for example, function `bint` below with the definition of *interleaving* given in §15.

```
def bint: Pr(Ac(L)) * Pr(Ac(L))  -> Pr(Ac(L))
    = (t1, t2) =>
      (| (r1,r2) => bh:
           union(taur(bh r1, r2), taul(bh r2, r1))
       |) (t1,t2).
```

where auxiliary functions `taur` and `taul` encode the right and left strengths, τ_r and τ_l, associated to the functor representing the process structure:

```
def taur: set(Ac(L) * B) * B -> set(Ac(L) * (B * B))
    = (s,t) => set{(a,x) => (a, (x,t))} s.

def taul: set(Ac(L) * B) * B -> set(Ac(L) * (B * B))
    = (s,t) => set{(a,x) => (a, (t,x))} s.
```

Similarly, *renaming* and *restriction* are encoded below as functions `bren` and `bret`, respectively. For convenience, the renaming homomorphism is represented as a mapping of type `map(Ac(Lb(N)),Ac(Lb(N)))`. Of course, this extends to a homomorphism by completing with the identity.

```
def bren{eqL: Lb(N) * Lb(N) -> bool}:
        Pr(Ac(Lb(N))) * map(Ac(Lb(N)),Ac(Lb(N)))
           -> Pr(Ac(Lb(N)))
    = (t, h) =>
      (| r => bh:
       set{x => { ff => x
                | ss a  => (a, p1 x)
                } app{eqA{eqL}}(compren h, p0 x)} (bh r)
       |) t.

def bret{eqL: Lb(N) * Lb(N) -> bool}:
        Pr(Ac(Lb(N))) * set(Lb(N))
           -> Pr(Ac(Lb(N)))
    = (t, k) =>
      (| r => bh:
         filter{x => not member{eqA{eqL}}(p0 x, compret k)}
         (bh r)
       |) t.
```

It remains to explain the meaning of functions `compret` and `compren`. First of all, note that the restriction set K of labels has to be extended to a set of actions, by application of the embedding `act`, before being used in `bret`. Additionally, it may also be ‘completed’ in order to cope with some syntactic conventions appearing in particular calculi. For example, to model CCS, it becomes necessary to close K with respect to label complement (*i.e.*, constructor `inv` in the CCS label algebra implementation given above). Both tasks are achieved by the function `compret`, which should be tuned to the syntactic particularities of the calculus considered. For the CCS example, it will look like

```
def compret: set(Lb(N)) -> set(Ac(Lb(N)))
    = s =>
      union( set{l => act(l)} s, set{l => act(inv(l))} s ).
```

Function `compren`, in the specification of `bren`, does a similar completion of the renaming homomorphism.

Finally, we consider prototyping *synchronous product* and *parallel*. Again the resulting CHARITY functions, `bsyn` and `bpar`, follow closely the formal definitions.

```
def bsyn{eqL: Lb(N) * Lb(N) -> bool}:
        Pr(Ac(Lb(N))) * Pr(Ac(Lb(N))) -> Pr(Ac(Lb(N)))
 = (t1, t2) =>
    (| (r1,r2) =>
         bh:  sel{eqL} deltar{eqL} (bh r1, bh r2)
     |) (t1,t2).

def bpar{eqL: Lb(N) * Lb(N) -> bool}:
        Pr(Ac(Lb(N))) * Pr(Ac(Lb(N))) -> Pr(Ac(Lb(N)))
 = (t1, t2) =>
       (| (r1,r2) =>
           bh: union( sel{eqL} deltar{eqL} (bh r1, bh r2),
                 union(taur(bh r1, r2), taul(bh r2, r1)) )
       |) (t1,t2).
```

where `deltar` and `sel`, implementing morphisms δ_r and sel, respectively, are defined as follows:

```
def sel{eqL: L * L -> bool}:
       set(Ac(L) * (B * B))  -> set(Ac(L) * (B * B))
 = s => filter{x => not eqA{eqL}(p0 x, nop)} s.

def deltar{eql: Lb(N) * Lb(N) -> bool}:
          set(Ac(Lb(N)) * B) * set(Ac(Lb(N)) * B)
          -> set(Ac(Lb(N)) * (B * B))
 = (l1, l2) =>
   set{(x,y) => (prodAc{eql}(p0 x, p0 y), (p1 x, p1 y))}
      (flatten set{ x => set{ y => (x,y) } l2 } l1).
```

42. ANIMATING PROCESSES. Once the specifications of process combinators have been translated into CHARITY, a functional implementation of a (family of) of calculi becomes available in which experiments can be carried out. By an experiment we mean that a process expression is supplied to the system and its evolution traced. In fact, all the allowed derivations are computed step by step, resorting to the evaluation mechanism of CHARITY (§E.5) for coinductive types. Animating processes is not essentially different from animating data oriented specifications in any of the rapid prototyping systems popular among the formal methods community, apart from the underlying shift to a coinductive setting. As a small example, consider the following CCS expression:

$$(a.b.\mathbf{0} \mid \overline{b}.\mathbf{0})\backslash_{\{b\}}$$

which is represented in the prototype by

```
bret{eqL{eq_string}}(
 bpar{eqL{eq_string}}
  (bpre(act(name("a")), bpre(act(name("b")), bnil)),
   bpre(act(inv(name("b"))), bnil)),  [name("b")]).
```

The notation is a bit verbose, namely because of parameterization requirements and the fact that all the embeddings (of identifiers into labels, through `name`, and of labels into actions, through `act`) are made explicit. In any case, at the level of a 'proof-of-concept', expressiveness, and not notation, is the key issue. CHARITY's evaluation of this expression is completed in three steps, as shown below. In each of them, the immediate derivatives of the process are computed, so that a progressively more complete picture of the derivation tree is revealed. The overall evaluation process is driven by the user, who can decide, at each derivation step, to pursue or stop execution:

```
(bh: ...)
Right display mode:
(q - quit, return - more) >>

(bh: [(act(name("a")), (bh: ...))])
Right display mode:
(q - quit, return - more) >>

(bh: [(act(name("a")), (bh: [(tau, (bh: ...))]))])
Right display mode:
(q - quit, return - more) >>

(bh: [(act(name("a")), (bh: [(tau, (bh: []))]))])
   : Pr(Ac(Lb(list(char))))
```

As the process denoted by this expression is finite, evaluation eventually terminates, the result of the last step representing the complete tree. Finite behaviour is, however, the exception, rather than the rule, in concurrent systems. But how can infinite behaviour be represented (and animated) in this setting? Such is the motivation for the following paragraphs.

43. RECURSIVE PROCESSES. As mentioned above, we have not yet any way of prototyping *recursive* processes, unless the dynamics of each particular example is supplied as a particular 'gene' coalgebra. This is, of course, an unsatisfactory solution.

The obvious way to deal with *recursive* processes in general consists of defining a *language* whose terms stand for process expressions, including a construction involving process *variables* as valid terms. Such variables should be bounded, in whatever one may take as the interpreter *environment*, by process equations of the form

$$v \;=\; exp$$

where v is a variable and *exp* a process expression. We have, however, to proceed with some care as it is well-known that not all defining equations determine process behaviour in an unique way. For example, any process is a solution to $v = v$ and equations like $v = v \,|\, v$ admit different, non bisimilar, solutions. One way of ensuring the existence of unique solutions, is to require that all variables occurring on the right hand side of an equation are guarded, in the sense that they are bounded by a prefix combinator. This has been proved in [Mil89] to be a sound criteria for the existence of unique solutions to (what is referred there as) strict bisimulation equations. Recall that this corresponds to equality in the final coalgebra. In fact, in [Mil89], guardedness is only required for variables wrt expressions in which they occurred. The extension, assumed hereafter, of this requirement to all variables in an expression does not appear to be a major restriction, while facilitating the development of the interpreter. Therefore, we shall consider in the process language a prefix-like construction — `pvar` — to introduce (guarded) variables in an expression.

Summing up we are left with the tasks of defining a term language for processes, its interpretation in the (final) semantic model and a suitable representation of an environment collecting the relevant process defining equations. Let us tackle these requirements one at a time.

44. A PROCESS LANGUAGE. As expected, a term language for processes, over a set `L` of labels, is defined as an inductive type. The CHARITY declaration below introduces `Ln(L)` as the initial algebra for a functor Σ induced by the following

BNF description:

$$\begin{array}{rllll} \langle \mathtt{P} \rangle ::= & \mathtt{pnil} & \mid \mathtt{ppre}\,(a, \langle \mathtt{P} \rangle) & \mid \mathtt{pcho}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) & \mid \\ & \mathtt{pint}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) & \mid \mathtt{psyn}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) & \mid \mathtt{ppar}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) & \mid \\ & \mathtt{pret}\,(\langle \mathtt{P} \rangle, K) & \mid \mathtt{pren}\,(\langle \mathtt{P} \rangle, f) & \mid \mathtt{pvar}\,(a, i) & \end{array}$$

where $a \in \mathtt{Ac(L)}$, $K \subseteq \mathtt{L}$, i is a process variable and f is a renaming Ac(L) homomorphism. Constructors pnil, ppre, pcho, pint, psyn, ppar, pret and pren correspond to the different process combinators. The only exception is pvar, which builds a new process given an action and a process variable. Its semantics will be later defined similarly to the one of the prefix combinator, according to the discussion in §43. The CHARITY declaration follows:

```
data Ln(L) -> P =
     pnil: 1 -> P | ppre: Ac(L) * P -> P |
     pcho: P * P -> P | pint: P * P -> P |
     psyn: P * P -> P | ppar: P * P -> P |
     pret: P * set(L) -> P |
     pren : P * map(Ac(L), Ac(L)) -> P |
     pvar: Ac(L) * string -> P.
```

Note that the definition is sufficiently generic, as it is parametric on L and resorts to whatever interaction structure is provided for Ac(L).

45. THE STRATEGY. How can Ln(L) expressions be interpreted as processes? Within the initial algebra approach to semantics, once fixed the syntax, a semantic Σ-algebra would be designed and the interpretation defined as the associated catamorphism. Our semantic universe, however, is the final coalgebra for functor $\mathcal{P}(\mathtt{Ac(L)} \times \mathsf{Id})$, and, therefore, the dual approach of final semantics comes in order. What has to be done is then to cast the syntax, *i.e.*, the set of terms Ln(L), into a $\mathcal{P}(\mathtt{Ac(L)} \times \mathsf{Id})$-coalgebra. The interpreter will now arise as the associated *anamorphism.*

Let function sem : Ln(L) $\longrightarrow$ Pr(Ac(L)) stand for the desired interpreter. The 'gene' for this anamorphism is a 'syntactic' coalgebra

$$\mathtt{syn} : \mathtt{Ln(L)} \longrightarrow \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Ln(L)})$$

which computes the 'syntactical' derivations of process terms. Observe, now, that the 'canonical' way to define syn is as a Σ-catamorphism. Its 'gene' is denoted by $\alpha_{\mathtt{syn}}$ in the sequel[1]. The diagram below contains the 'full' picture. Notice that $\alpha_{\mathtt{syn}}$ is

[1] It may be instructive to pay some attention to the behaviour of function $\alpha_{\mathtt{syn}}$. Let a, b be actions and s, t process expressions (*i.e.*, inhabitants of Ln(L)). Then, $\alpha_{\mathtt{syn}}$ will map $\mathtt{pcho}\,(\{\langle a, s\rangle\}, \{\langle b, t\rangle\})$ to $\{\langle a, s\rangle, \langle b, t\rangle\}$. Similarly, $\mathtt{psyn}\,(\{\langle a, s\rangle\}, \{\langle b, t\rangle\})$ will be mapped to set $\{\langle x, \mathtt{psyn}\,(s, t)\rangle \mid x$ in action product of a and b, once filtered synchronisation failures$\}$.

actually the *only* function to be supplied by the programmer. Then, we get for free

$$\mathtt{syn} \;=\; (\!|\alpha_{\mathtt{syn}}|\!)_{\Sigma}$$

and

$$\mathtt{sem} \;=\; [\!(\mathtt{syn})\!]_{\mathcal{P}(\mathtt{Ac\,(L)}\times \mathsf{Id})}$$

Thus,

$$\begin{tikzcd}
\mathtt{Pr\,(Ac\,(L))} \arrow[r, "\omega"] & \mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Pr\,(Ac\,(L))}) \\
\mathtt{Ln\,(L)} \arrow[u, "\mathtt{sem}"] \arrow[r, "\mathtt{syn}"] & \mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)}) \arrow[u, "\mathcal{P}(\mathsf{id}\times\mathtt{sem})"'] \\
\Sigma\ \mathtt{Ln\,(L)} \arrow[u, "\alpha"] \arrow[r, "\Sigma\ \mathtt{syn}"] & \Sigma\ \mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)}) \arrow[u, "\alpha_{\mathtt{syn}}"']
\end{tikzcd}$$

where ω and α are, respectively, the final $\mathcal{P}(\mathtt{Ac\,(L)} \times \mathsf{Id})$-coalgebra and the initial Σ-algebra.

46. ENVIRONMENT. Our last question concerns the introduction of an environment to the interpreter in order to collect all the process defining equations relevant to drive experiments on a particular network of processes. Such an environment E can be thought of as a mapping assigning to each process variable an expression in $\mathtt{Ln\,(L)}$. Assuming variable identifiers are modeled by strings, E will be typed in CHARITY as map(string, Ln(L)). Clearly, E acts as a supplier of context information to the interpretation function sem, which becomes typed as

$$\mathtt{sem} : \mathtt{Ln\,(L)} \times E \longrightarrow \mathtt{Pr\,(Ac\,(L))}$$

All types in CHARITY are *strong* and, therefore, this extra parameter is smoothly accommodated in the framework. In fact, both sem and syn become defined as, respectively, a *strong unfold* (§3.56) for $\mathcal{P}(\mathtt{Ac\,(L)} \times \mathsf{Id})$ and a *strong fold* (§3.61) for Σ. Our previous diagram remains valid, but has to be interpreted in the Kleisli category for the *product comonad*. Its interpretation in the original category, along the lines discussed in chapter 3, is depicted in the diagram below, which makes the structure involved explicit. Notice that ω' and α' are, respectively, the corresponding final coalgebra and initial algebra in the Kleisli, defined simply as $\omega \cdot \pi_1$ and $\alpha \cdot \pi_1$ as explained in §3.59. Thus,

$$\begin{tikzcd}
\mathtt{Pr(Ac(L))} \times E \arrow[rrr, "\omega'"] \arrow[d, dotted, no head] & & & \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Pr(Ac(L))}) \\
\mathtt{Pr(Ac(L))} & & & \\
\mathtt{Ln(L)} \times E \arrow[u, "\mathtt{sem}"] \arrow[rr, "\mathtt{syn}"] \arrow[d, dotted, no head] & & \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Ln(L)}) \arrow[r, dotted, no head] & \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Ln(L)}) \times E \arrow[uu] \\
\mathtt{Ln(L)} & & & \\
\Sigma\ \mathtt{Ln(L)} \times E \arrow[u, "\alpha'"] \arrow[r] & \Sigma\ \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Ln(L)}) \arrow[r, dotted, no head] & \Sigma\ \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Ln(L)}) \times E \arrow[uu, "\alpha_{\mathtt{syn}}"'] &
\end{tikzcd}$$

Formally, the interpretation function arises as

$$\mathtt{sem} = \mathsf{unfold}_{\mathcal{P}(\mathtt{Ac(L)} \times \mathsf{Id})}\ \mathtt{syn}$$

where

$$\mathtt{syn} = \mathsf{fold}_{\Sigma}\ \alpha_{\mathtt{syn}}$$

More precisely, as the definition of `syn` is made in terms of both the computations on the substructures of its argument and these substructures themselves, it arises in fact as a *strong paramorphism* (§E.10). This is simply implemented by the CHARITY fold combinator and the annotation #, as explained in §E.10.

47. INTERPRETATION. We are ready to present the actual code for the interpretation function `sem`. Not much remains to be said about this definition, as the encoding of each process combinator has already been detailed in the beginning of this section. The interpretation of the new construction `pvar(a,i)` is as expected: the continuation process arises as the interpretation of the process expression associated to variable `i`, if `i` is collected in the environment, or `pnil` otherwise. Thus,

```
def sem{eqL: Lb(N) * Lb(N) -> bool}:
        Ln(Lb(N)) * map(string, Ln(Lb(N)))  -> Pr(Ac(Lb(N)))
    = (exp,m)  => (| e => bh: syn{eqL}(e,m) |) exp.
```

where

```
def syn{eqL: Lb(N) * Lb(N) -> bool}:
     Ln(Lb(N)) * map(string, Ln(Lb(N)))
     ->  set(Ac(Lb(N)) * Ln(Lb(N)))

 = (pr,m) =>
   {| pnil: ()      => empty
    | ppre: (a,l)   => sing(a, p1 #)
    | pvar: (a,s)   => { ss(p) => sing(a, p)
                       | ff    => empty
                       } app{eq_string}(m,s)
    | pcho: (lp,lq) => union(lp, lq)
    | pint: (lp,lq) =>
            union(stau1(lp, p1 #), stau1(lq,  p0 #))
    | psyn: (lp,lq) =>
            ssel{eqL} sdelta1{eqL} (lp,lq)
    | ppar: (lp,lq) =>
            union(union(stau2(lp, p1 #), stau2(lq,  p0 #)),
                  ssel{eqL} sdelta2{eqL} (lp,lq))
    | pret: (l,k)   =>
            set{x => (p0 x, pret(p1 x, k))}
            filter{x => not member{eqA{eqL}}
                   (p0 x, compren k)} l
    | pren: (l,h)   =>
            set{x => { ff => (p0 x,  pren(p1 x, h))
                     | ss a  => (a, pren(p1 x, h))
                     } app{eqA{eqL}}(compret h, p0 x) } l
    |} pr.
```

To fully understand the definition above, observe that the derivations of a process expression are (a set of pairs of actions and) *process expressions*, whereas, in the previous definition of 'stand-alone' combinators, they were defined in terms of *processes* themselves. As an illustration, compare the entry corresponding to renaming in the 'gene' of `syn` with the definition of `bren` in §40. The same observation justifies the following auxiliary definitions of `stau1`, `stau2`, `ssel`, `sdelta1` and `sdelta2`, whose role is similar to the original `taur`, `sel` and `deltar` functions in §40.

```
def stau1: set(Ac(L) * Ln(L)) * Ln(L) -> set(Ac(L) * Ln(L))
    = (s,p) => set{(a,x) => (a, pint(x,p))} s.

def stau2: set(Ac(L) * Ln(L)) * Ln(L) -> set(Ac(L) * Ln(L))
    = (s,p) => set{(a,x) => (a, ppar(x,p))} s.
```

```
def ssel{eql: L * L -> bool}: set(Ac(L) * B)
    -> set(Ac(L) * B)
    = s => filter{x => not eqA{eql}(p0 x, nop)} s.

def sdelta1{eql: Lb(N) * Lb(N) -> bool}:
    set(Ac(Lb(N)) * Ln(Lb(N))) * set(Ac(Lb(N)) * Ln(Lb(N)))
    -> set(Ac(Lb(N)) * Ln(Lb(N)))
 = (l1, l2) =>
   set{(x,y) => (prodAc{eql}(p0 x, p0 y), psyn(p1 x, p1 y))}
      (flatten set{ x => set{ y => (x,y) } l2 } l1).

def sdelta2{eql: Lb(N) * Lb(N) -> bool}:
    set(Ac(Lb(N)) * Ln(Lb(N))) * set(Ac(Lb(N)) * Ln(Lb(N)))
   -> set(Ac(Lb(N)) * Ln(Lb(N)))
 = (l1, l2) =>
   set{(x,y) => (prodAc{eql}(p0 x, p0 y), ppar(p1 x, p1 y))}
      (flatten set{ x => set{ y => (x,y) } l2 } l1).
```

48. PROTOTYPING RECURSIVE PROCESSES. Consider the following example, due to C. Stirling [Sti95], of a CCS process describing a controller for a road-railway junction. Actions car and $train$ model, respectively, a car and a train approaching the junction. The system is controlled by a process $Signal$ which enforces a strict sequencing of the control signs that allow or forbid traffic in each branch of the junction. The latter are associated to a traffic light with actions $green$ and red and the rail barrier responding to actions up and dw. Finally, actions $\overline{ccross}$ and $\overline{tcross}$ are supplied (externally) by sensors in the junction to communicate that a car, in the first case, and a train, in the second, have effectively crossed the junction.

$$\begin{aligned} \mathit{Road} &\triangleq car.up.\overline{ccross}.\overline{dw}.\mathit{Road} \\ \mathit{Rail} &\triangleq train.green.\overline{tcross}.\overline{red}.\mathit{Rail} \\ \mathit{Signal} &\triangleq \overline{green}.red.\mathit{Signal} + \overline{up}.dw.\mathit{Signal} \end{aligned}$$

$$C \triangleq (\mathit{Road} \mid \mathit{Rail} \mid \mathit{Signal})\backslash_{\{green,red,up,dw\}}$$

The whole junction is modeled by a composition of three processes each of which is separately defined. Therefore, the system is represented in the prototyper (suitably instantiated with the CCS interaction structure) as shown below.

```
sem{eqL{eq_string}}(
 pret( ppar( ppar(pvar(act(name("iR")),"ROAD"),
                  pvar(act(name("iT")),"TRAIN")),
                  pvar(act(name("iS")),"SIGNAL") )
   [name("up"), name("green"), name("red"), name("dw")]),
  [("ROAD",   ppre(act(name("car")), ppre(act(name("up")),
              ppre(act(inv(name("ccross"))),
              pvar(act(inv(name("dw"))), "ROAD"))))),
   ("TRAIN",  ppre(act(name("train")),
              ppre(act(name("green")),
              ppre(act(inv(name("tcross"))),
              pvar(act(inv(name("red"))), "TRAIN"))))),
   ("SIGNAL", pcho(
              ppre(act(inv(name("green"))),
                  pvar(act(name("red")), "SIGNAL")),
              ppre(act(inv(name("up"))),
                  pvar(act(name("dw")), "SIGNAL")) )) ]).
```

Processes $Road$, $Rail$ and $Signal$, along with the respective defining equations, are grouped in the environment for C. We have, however, to deal with a syntactic restriction of our process language. In fact, process variables in `Ln(L)` always appear within a `pvar` constructor and, therefore, 'dummy' initial actions were introduced. Conceptually, we may think about them as 'triggers' of the associated processes. This is not, however, a fundamental restriction of the approach, as other design solutions would have been possible for `Ln(L)`. For example, we could have allowed process variables to occur freely in an expression, provided that process expressions in the environment were tested for guardedness and their initial actions properly computed. The adopted solution seems reasonable for illustration purposes and very simple to implement.

49. REMARK. The CHARITY implementation of an interpreter for `Ln(L)` assumes an effective representation of sets and set-theoretic operators. In practice, however, our implementation of sets is based on sequences, a common trick in functional languages. This fact may introduce some undesirable behaviour in the prototype. The problem arises in the use of set `union`, whose implementation resorts to sequence concatenation. Being implemented by an operator on an inductive type, `union` is evaluated eagerly. So, suppose a parallel composition of a finite with an infinite process is being computed. When the evaluation of the first argument to the underlying `union` ends, the CHARITY machine takes the *current* evaluation of the second, which correspond to a partial result, and terminates! The problem manifests itself whenever the evaluation of a first argument to a `union` terminates. As is easily foreseen, this may happen in several situations: not only when the corresponding process is finite, but also as a result of a restriction enforcing the death of the process.

A solution to this problem will have to provide a better representation for sets, but this seems difficult to implement. For example, the coinductive representation of sets by their characteristic functions, discussed in appendix E, cannot be used here as CHARITY does not allow the 'state variable' in a coinductive declaration to occur contravariantly. The declaration of `Pr(A)` would become invalid in first place.

An alternative solution, which we have implemented, consists in extending all terminating processes by an infinite sequence of a special action `dead`. The derivation tree is completed by appending to each leaf such a linear special branch. This does not introduce any semantic problem — we are just adopting a different representation for inaction! — and actually solves the problem by avoiding termination in the evaluation of `union`.

From a programming point of view, the solution involves a minor change in the interpreter: in the code for `syn` the entries corresponding to `pnil`, `pvar` and `pret` are modified as follows:

```
...
   | pnil: ()      => sing(dead,pnil)
   | pvar: (a,s)   => { ss(p) => sing(a, p)
                      | ff    => sing(dead,pnil)
                      } app{eq_string}(m,s)
...
   | pret: (l,k)   =>
       set{x => { true  => (p0 x, pret(p1 x, k))
                | false => (dead, pnil)
                } not member{eqA{eqL}}(p0 x, complren k)
...
```

Notice that in the first two cases `empty` is replaced by the derivation `sing(dead, pnil)`. In the case of restriction, the forbidden derivations are also replaced by `sing(dead, pnil)`, instead of being simply omitted. This enforces replacement of the filter construction (used previously) by an explicit transformation of each element of the derivation set.

The only disadvantage of this solution is the eventual proliferation of `dead` symbols along the evaluation output, which may decrease readability of such a (already verbose) text. As commented in chapter 7, a more elaborated prototyping kernel for processes, currently under development, incorporates some editing functions applicable to CHARITY outputs which, besides providing a pretty-printer functionality, do filter, in each evaluation step, the unpleasant `dead` symbols, thus restoring the original derivation trees (or their successive approximations).

4. Some Variants

50. PROCESSES. In the beginning of this chapter it was remarked that the proposed approach to process calculi design could cope with a variety of particular cases, because the emphasis was placed on the common underlying structures rather than on the distinctive particularities. A first, major, source of *genericity* has already been introduced by separating the *behaviour* from the *interaction* structures. Note that all the process combinators introduced are either *independent* of any particular interaction discipline or *parametrized* by it.

In this section we shall briefly examine what happens in case the behaviour structure itself is changed. Recall that processes were previously defined as inhabitants of the carrier of the final coalgebra for

$$\mathsf{T} \;=\; \mathsf{B}\;(Act \times \mathsf{Id})$$

where B was taken as the finite powerset functor. Later on we have shown that, assuming a monoidal structure over *Act*, the behaviour model captured by $\mathcal{P}(Act \times \mathsf{Id})$ becomes a *strong monad*. The definitions of some combinators build upon this and, in particular, *synchronous product* relies on the monad distribution law $\delta\!\!\!\tau$. Commutativity of $\otimes$, and consequently of $|$, depends on the monoid underlying the interaction structure being itself Abelian. Therefore, a first line of enquire followed in the sequel consists of replacing B by different monads and extracting corresponding families of calculi still parametrized by the interaction structure.

51. PARTIAL PROCESSES. The simplest case takes B as the identity Id. The result is, of course, a universe of *deterministic* (and perpetual) processes. We shall investigate a further elaboration of this, replacing Id by $\mathsf{Id} + \mathbf{1}$. Therefore, processes, as elements of the carrier of the final coalgebra for

$$\mathsf{T} \;=\; (Act \times \mathsf{Id}) + \mathbf{1}$$

are deterministic but also *partial* in the sense that derivation to a 'dead' state is always possible. Such a process universe is introduced in CHARITY through the following declaration:

```
data C -> PartialPr(A) = bh: C -> SF(A * C).
```

The resulting calculus is far less expressive than the one previously discussed. In fact derivations do not form any kind of *collection* and, therefore, nondeterminism is ruled out. Similarly, combinators which explore nondeterminism lack a counterpart here. Such is the case of *choice*, *interleaving* and, as a generalisation of the latter, *parallel*. On the other hand, the composition of $\mathsf{Id} + \mathbf{1}$ with the monoidal monad generated by *Act* is still a strong Abelian monad, and therefore *synchronous product* is still definable along the lines of §28. Let us explore what we have been left with.

52. Dynamic Combinators. *Inaction* and *prefix* are defined as

$$\omega \cdot \mathsf{nil} \;=\; \iota_2$$

and

$$\omega \cdot a. \;=\; \iota_1 \cdot \mathsf{label}_a$$

where label_a is as before. Choice, as explained above, is ruled out. We may, however, consider a *deterministic* choice combinator — $+_d$ — which gives precedence to, say, the first argument. This is to say that the derivation of $p +_d q$ would be the derivation of p but if p, but not q, becomes inert. Formally,

$$\begin{aligned} \omega \cdot +_d \;&=\; \nu \times \nu \xrightarrow{\omega \times \omega} (Act \times \nu + \mathbf{1}) \times (Act \times \nu + \mathbf{1}) \\ &\xrightarrow{\mathsf{dl}} ((Act \times \nu) \times (Act \times \nu + \mathbf{1})) + (\mathbf{1} \times (Act \times \nu + \mathbf{1})) \\ &\xrightarrow{(\mathsf{id}+\mathsf{l})\cdot\mathsf{dr}+\mathsf{r}} ((Act \times \nu) \times (Act \times \nu) + (Act \times \nu)) + (Act \times \nu + \mathbf{1}) \\ &\xrightarrow{[[\iota_1\cdot\pi_1,\iota_1],\mathsf{id}]} Act \times \nu + \mathbf{1} \end{aligned}$$

The combinator is, of course, non commutative.

53. Static Combinators. On the other hand, *product*, *restriction* and *renaming* can be defined in a rather generic way as anamorphisms whose ‘genes’ are parametrized by the monad B:

$$\begin{aligned} \alpha_{[f]} \;&=\; \nu \xrightarrow{\omega} \mathsf{B}\;(Act \times \nu) \xrightarrow{\mathsf{B}\;(f\times\mathsf{id})} \mathsf{B}\;(Act \times \nu) \\ \alpha_{\backslash_K} \;&=\; \nu \xrightarrow{\omega} \mathsf{B}\;(Act \times \nu) \xrightarrow{\mathsf{filter}_K{}^{\mathsf{B}}} \mathsf{B}\;(Act \times \nu) \\ \alpha_{\otimes} \;&=\; \nu \times \nu \xrightarrow{(\omega\times\omega)} \mathsf{B}\;(Act \times \nu) \times \mathsf{B}\;(Act \times \nu) \xrightarrow{\delta_r^{\mathsf{B}\;(Act\times\mathsf{Id})}} \mathsf{B}\;(Act \times (\nu \times \nu)) \\ &\xrightarrow{\mathsf{sel}^{\mathsf{B}}} \mathsf{B}\;(Act \times (\nu \times \nu)) \end{aligned}$$

where $\delta_r^{\mathsf{B}\;(Act\times\mathsf{Id})}$ is the distribution law associated to the composed monad $\mathsf{B}\;(Act \times \mathsf{Id})$, therefore encapsulating the θ operation on actions. On the other hand, $\mathsf{sel}^{\mathsf{B}}$ and $\mathsf{filter}_K{}^{\mathsf{B}}$ explore the B structure in order to rule out synchronisation failures, in the first case, and to perform the action restriction in the second. For $\mathsf{B} = Act \times \mathsf{Id} + \mathbf{1}$,

$$\mathsf{filter}_K{}^{\mathsf{Id}+\mathbf{1}} \;=\; [((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2]$$

is expressed by a conditional, whereas, for $\mathsf{B} = \mathcal{P}$, such a conditional was iterated over a set. Again $\mathsf{sel}^{\mathsf{Id}+\mathbf{1}} = \mathsf{filter}_{\{0\}}{}^{\mathsf{Id}+\mathbf{1}}$.

Proofs of properties of partial processes also follow the structure and style used before. As an illustration, we shall prove here the following lemma, which corresponds to the result in §19.

54. LEMMA. For any $K \subseteq L$, $\backslash_K \cdot \backslash_K = \backslash_K$.

Proof.

$$\begin{aligned}
& \backslash_K \cdot \backslash_K \;=\; \backslash_K \\
\equiv \quad & \qquad \{ \text{ definition } \} \\
& \llbracket \alpha_{\backslash_K} \rrbracket \cdot \backslash_K \;=\; \llbracket \alpha_{\backslash_K} \rrbracket \\
\Leftarrow \quad & \qquad \{ \text{ ana fusion (3.9) } \} \\
& \alpha_{\backslash_K} \cdot \backslash_K \;=\; ((\mathsf{id} \times \backslash_K) + \mathbf{1}) \cdot \alpha_{\backslash_K}
\end{aligned}$$

and this equality holds because

$$\begin{aligned}
& \alpha_{\backslash_K} \cdot \backslash_K \\
= \quad & \qquad \{ \; \alpha_{\backslash_K} \text{ definition } \} \\
& [((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2] \cdot \omega \cdot \backslash_K \\
= \quad & \qquad \{ \text{ comorphism } \} \\
& [((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2] \cdot ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot [(\in_K \cdot\pi_1 \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2] \cdot \omega \\
= \quad & \qquad \{ \; + \text{ fusion } \} \\
& [((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2] \\
& \cdot[((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot ((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot \iota_2] \cdot \omega \\
= \quad & \qquad \{ \text{ conditional (2.41), } + \text{ cancellation } \} \\
& [((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2] \\
& \cdot[((\in_K \cdot\pi_1) \;\rightarrow\; ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot \iota_2\cdot!,\; ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot \iota_1), \iota_2] \cdot \omega \\
= \quad & \qquad \{ \; + \text{ cancellation } \} \\
& [((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2] \cdot [((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1 \cdot (\mathsf{id} \times \backslash_K)), \iota_2] \cdot \omega \\
= \quad & \qquad \{ \; + \text{ fusion, } + \text{ cancellation } \} \\
& [[((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1), \iota_2] \cdot ((\in_K \cdot\pi_1) \;\rightarrow\; \iota_2\cdot!,\, \iota_1 \cdot (\mathsf{id} \times \backslash_K)), \iota_2] \cdot \omega
\end{aligned}$$

$$
\begin{array}{cl}
= & \qquad \{ \text{ conditional (2.43), + cancellation } \} \\
 & [((\in_K \cdot \pi_1) \;\rightarrow\; \iota_2 \cdot !),\ \iota_1 \cdot (\mathsf{id} \times \backslash_K)), \iota_2] \cdot \omega \\
= & \qquad \{ \text{ + cancellation } \} \\
 & [((\in_K \cdot \pi_1) \;\rightarrow\; ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot \iota_2 \cdot !,\ ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot \iota_1), \iota_2] \cdot \omega \\
= & \qquad \{ \text{ conditional (2.41), + cancellation } \} \\
 & ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot [((\in_K \cdot \pi_1) \;\rightarrow\; \iota_2 \cdot !,\ \iota_1), \iota_2] \cdot \omega \\
= & \qquad \{ \ \alpha_{\backslash_K} \text{ definition } \} \\
 & ((\mathsf{id} \times \backslash_K) + \mathsf{id}) \cdot \alpha_{\backslash_K}
\end{array}
$$

□

55. ORDERED NO DETERMINISM. Another possibility for B is the type functor associated to the sequence data type. The resulting process universe is then the final coalgebra for

$$\mathsf{T} \;=\; (Act \times \mathsf{Id})^*$$

which is declared in CHARITY as follows:

```
   data C -> OrderedPr(A) = bh: C -> list(A * C).
```

In this case there exists, for each process, a collection of derivations, thus expressing a form of nondeterminism. Such collection is, however, modelled by a sequence which imposes an order (*e.g.*, of probability, cost, etc.) on them. Clearly Id^* is a strong monad and its composition with $(Act \times \mathsf{Id})$ is well defined and also strong. However, the resulting monad is not commutative, even when the interaction structure is an Abelian monoid. As a consequence, commutativity is lost in several combinators and, in particular, we get two non bisimilar versions of *product*, based, respectively, in δ_r and δ_l. Recall, from §A.10, that equation $\delta_r \;=\; \delta_l$ holds only for commutative monads.

56. REACTIVE PROCESSES. So far we have been working under what has been labelled in §8 the *'active' interpretation* of processes. In the same paragraph, however, an alternative interpretation was suggested in which the behaviour of a process results from reaction to external stimuli rather than from commitment into actions. Such a distinction, which is often left implicit in the literature on process calculi, can be easily accommodated in the approach we have been discussing. Formally, an universe for reactive processes is specified as a final coalgebra $\overline{\omega}$ for

$$\mathsf{T} \;=\; (\mathsf{B}\ \mathsf{Id})^{Act}$$

where B captures, as usual, the underlying behaviour structure.

In the remaining of this section we shall revisit the process combinators in this new setting, taking as B the finite powerset monad. Of course, all the variants for B discussed above would do as well and the remarks on the expressiveness of the resulting calculi remain valid.

57. DYNAMIC COMBINATORS. The definitions of *inaction*, *prefix* and *choice* follow closely the ones already considered for 'active' processes, reflecting, however, the fact that *Act* appears now as an exponent. Note, for example, how prefixing a process p by an action a results in a new process which is blind for every stimulus different from a. Under the 'active' interpretation such a process appears with a singular set of derivations witnessing commitment to a. Formally,

$$\begin{aligned} \overline{\omega} \cdot \mathsf{nil} &= \overline{\underline{\emptyset} \cdot !} \\ \overline{\omega} \cdot a. &= \overline{((=_a \cdot \pi_2) \to \mathsf{sing} \cdot \pi_1,\ \underline{\emptyset} \cdot !)} \end{aligned}$$

for any $a \in Act$, and

$$\overline{\omega} \cdot + = \overline{\cup \cdot (\omega \times \omega) \cdot \mathsf{pdl}}$$

where $\mathsf{pdl} : (X \times Y) \times Z \longrightarrow (X \times Z) \times (Y \times Z)$ is defined as $\langle \pi_1 \times \mathsf{id}, \pi_2 \times \mathsf{id} \rangle$.

58. RESTRICTION AND RENAMING. For any $K \subseteq L$ and renaming homomorphism f, combinators modelling *restriction* and *renaming* are given by

$$\backslash_K = [\!(\overline{\alpha_K})\!] \qquad \text{where} \qquad \alpha_K = ((\notin_K \cdot \pi_2) \to \omega,\ \underline{\emptyset} \cdot !)$$

and

$$[f] = [\!(\overline{\alpha_f})\!] \qquad \text{where} \qquad \alpha_f = \omega \cdot (\mathsf{id} \times f)$$

Note, once again, that both f and the restriction set K act by constraining the set of meaningful stimuli before the effective derivation is computed. Recall that, under the 'active' interpretation, their effect was defined over the (previously computed) derivations (§17 and §22).

59. PARALLEL COMPOSITION. *Synchronous product* is defined, in this setting, as

$$\otimes = [\!(\alpha_\otimes)\!]$$

where

$$\begin{aligned} \alpha_{\otimes} \;=\; & \nu \times \nu \xrightarrow{(\overline{\omega} \times \overline{\omega})} \mathcal{P}\nu^{Act} \times \mathcal{P}\nu^{Act} \xrightarrow{\mathsf{prod}} \mathcal{P}(\mathcal{P}\nu \times \mathcal{P}\nu)^{Act} \\ & \xrightarrow{(\mathcal{P}\delta_r^{\mathcal{P}})^{Act}} \mathcal{PP}(\nu \times \nu)^{Act} \xrightarrow{\cup^{Act}} \mathcal{P}(\nu \times \nu)^{Act} \end{aligned}$$

where

$$\mathsf{prod}\ \langle d_1, d_2 \rangle \;=\; \lambda a\,.\ (a = 0 \rightarrow \emptyset,\ \{\langle d_1\ a_1, d_2\ a_2 \rangle |\ \forall_{a_1, a_2 \in Act}\ .\ a = a_1 \theta a_2\})$$

Recall from §28, that, in the ‘active’ case, interaction was neatly captured by the distribution law for the $\mathcal{P}(Act \times \mathsf{Id})$ monad, which is no longer the case here. This entails the need to introduce function prod. Additionally, prod filters out synchronisation failures and thus replaces sel.

On the other hand, the definition of the *interleaving* combinator matches with the one for the ‘active’ case, but requires the replacement of set union by

$$\mathsf{merge}\ \langle d_1, d_2 \rangle \;=\; \lambda a.\ d_1\ a \cup d_2\ a$$

Thus, $||| = [\![(\alpha_{|||})]\!]$ with

$$\begin{aligned} \alpha_{|||} \;=\; & \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\overline{\omega} \times \mathsf{id}) \times (\mathsf{id} \times \overline{\omega})} (\mathcal{P}\nu^{Act} \times \nu) \times (\nu \times \mathcal{P}\nu^{Act}) \\ & \xrightarrow{\tau_r \times \tau_l} \mathcal{P}(\nu \times \nu)^{Act} \times \mathcal{P}(\nu \times \nu)^{Act} \xrightarrow{\mathsf{merge}} \mathcal{P}(\nu \times \nu)^{Act} \end{aligned}$$

where, of course, the right and left strengths are relative to the $(\mathcal{P}\mathsf{Id})^{Act}$ monad. The same observation applies to *parallel* composition, which arises, once again, as a combination of product and interleaving at ‘genes’ level. Comparing with the definition in §32, set union is, again, replaced by merge. Formally,

$$| \;=\; [\![(\alpha_{|})]\!]$$

where

$$\begin{aligned} \alpha_{|} \;=\; & \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\alpha_{|||} \times \alpha_{\otimes})} \mathcal{P}(\nu \times \nu)^{Act} \times \mathcal{P}(\nu \times \nu)^{Act} \\ & \xrightarrow{\mathsf{merge}} \mathcal{P}(\nu \times \nu)^{Act} \end{aligned}$$

60. PROPERTIES. The basic properties of this model for reactive processes coincide with the properties of the corresponding calculus of ‘active’ processes developed in section 2. The proof style is also similar, although calculation resorts now heavily on the properties of exponentials (§2.33). As an illustration, we prove below the result corresponding to law (4.5).

61. LEMMA. For any $K \subseteq L$,

$$\backslash_K \cdot + \;=\; + \cdot (\backslash_K \times \backslash_K)$$

Proof.

$$\begin{aligned}
& \overline{\omega} \cdot \backslash_K \cdot + \\
= \quad & \qquad \{ \text{ comorphism } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{\alpha_K} \cdot + \\
= \quad & \qquad \{ \ \alpha_K \text{ definition } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{((\in_K \cdot\pi_2) \to \omega,\, \underline{\emptyset}\cdot!)} \cdot + \\
= \quad & \qquad \{ \text{ exponential fusion } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{((\in_K \cdot\pi_2) \to \omega,\, \underline{\emptyset}\cdot!) \cdot (+ \times \mathsf{id})} \\
= \quad & \qquad \{ \text{ conditional (2.42) } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{((\in_K \cdot\pi_2 \cdot (+ \times \mathsf{id})) \to \omega \cdot (+ \times \mathsf{id}),\, \underline{\emptyset}\cdot! \cdot (+ \times \mathsf{id}))} \\
= \quad & \qquad \{ \ + \text{ and ! definitions } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{((\in_K \cdot\pi_2) \to \cup \cdot (\omega \times \omega) \cdot \mathsf{pdl},\, \underline{\emptyset}\cdot!)} \\
= \quad & \qquad \{ \ \cup \text{ and ! definitions } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{((\in_K \cdot\pi_2) \to \cup \cdot (\omega \times \omega) \cdot \mathsf{pdl},\, \cup \cdot ((\underline{\emptyset}\cdot!) \times (\underline{\emptyset}\cdot!)) \cdot \mathsf{pdl})} \\
= \quad & \qquad \{ \text{ conditional (2.41) } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{\cup \cdot ((\in_K \cdot\pi_2) \to (\omega \times \omega) \cdot \mathsf{pdl},\, ((\underline{\emptyset}\cdot!) \times (\underline{\emptyset}\cdot!)) \cdot \mathsf{pdl})} \\
= \quad & \qquad \{ \ \pi_2 = \pi_2 \cdot \pi_2 \cdot \mathsf{pdl} \text{ and conditional (2.42) } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{\cup \cdot ((\in_K \cdot\pi_2 \cdot \pi_2) \to (\omega \times \omega),\, ((\underline{\emptyset}\cdot!) \times (\underline{\emptyset}\cdot!))) \cdot \mathsf{pdl}} \\
= \quad & \qquad \{ \text{ conditional distribution over } \times \text{ and } \mathsf{pdl} \text{ definition } \} \\
& \mathcal{P}\backslash_K{}^{Act} \cdot \overline{\cup \cdot (((\in_K \cdot\pi_2) \to \omega,\, (\underline{\emptyset}\cdot!)) \times ((\in_K \cdot\pi_2) \to \omega,\, (\underline{\emptyset}\cdot!))) \cdot \mathsf{pdl}} \\
= \quad & \qquad \{ \text{ exponential absorption, } \alpha_K \text{ definition } \} \\
& \overline{\mathcal{P}\backslash_K \cdot \cup \cdot (\alpha_K \times \alpha_K) \cdot \mathsf{pdl}} \\
= \quad & \qquad \{ \ \cup \text{ natural } \}
\end{aligned}$$

$$
\begin{aligned}
& \overline{\cup \cdot (\mathcal{P}\backslash_K \times \mathcal{P}\backslash_K) \cdot (\alpha_K \times \alpha_K) \cdot \mathsf{pdl}} \\
= & \qquad \{ \ \times \text{ functor} \ \} \\
& \overline{\cup \cdot (\mathcal{P}\backslash_K \cdot \alpha_K \times \mathcal{P}\backslash_K \cdot \alpha_K) \cdot \mathsf{pdl}} \\
= & \qquad \{ \text{ comorphism } \} \\
& \overline{\cup \cdot ((\omega \cdot (\backslash_K \times \mathsf{id})) \times (\omega \cdot (\backslash_K \times \mathsf{id}))) \cdot \mathsf{pdl}} \\
= & \qquad \{ \ \times \text{ functor} \ \} \\
& \overline{\cup \cdot (\omega \times \omega) \cdot ((\backslash_K \times \mathsf{id}) \times (\backslash_K \times \mathsf{id})) \cdot \mathsf{pdl}} \\
= & \qquad \{ \ \mathsf{pdl} \text{ natural} \ \} \\
& \overline{\cup \cdot (\omega \times \omega) \cdot \mathsf{pdl} \cdot ((\backslash_K \times \backslash_K) \times \mathsf{id})} \\
= & \qquad \{ \text{ exponential fusion } \} \\
& \overline{\cup \cdot (\omega \times \omega) \cdot \mathsf{pdl}} \cdot (\backslash_K \times \backslash_K) \\
= & \qquad \{ \ + \text{ definition} \ \} \\
& \overline{\omega} \cdot + \cdot (\backslash_K \times \backslash_K)
\end{aligned}
$$

□

5. From Processes to Components

62. COMPONENTS. In chapter 1 software components have been characterised as dynamic systems with a public interface and a private encapsulated state. The relevance of state information precludes a ‘process-like’ view of components, as inhabitants of a final coalgebra. Components are themselves *concrete* coalgebras. For each value of the state space, a corresponding ‘process’, or *behaviour*, arises by computing its anamorphic image. Components do have, however an underlying behaviour model built in their ‘shapes’. Therefore, and with respect to the simple systems taxonomy discussed in section 1, we shall

- consider the simultaneous presence of an input and an output observation universes, whose types will be referred to as the *interface types*;
- abstract away from a particular behavioural model (like determinism or non-determinism), by parameterizing the signature functor specification by a *strong monad* intended to capture such model.

63. SHAPES. Keeping in mind the requirements above, components will be modelled as concrete coalgebras with specified initial conditions (typically a distinguished initial state, referred to as the *seed* value). Our first step is then to choose a family of functors to suitably capture components' interfaces. Such is the topic of the present section, which paves the way to the development of component calculi in the following chapters. Our starting point is the following family of Set endofunctors:

$$\mathsf{T}^{\mathsf{B}} \;=\; O'^{I'} \times \mathsf{B}(\mathsf{Id} \times O)^{I}$$

where the sets I, I' and O, O' are, respectively, the input and output observation universes which ensure the flow of data. On the other hand, B is a strong monad intended to capture a particular behaviour model (see §64 later on).

Each T^{B}-coalgebra p over carrier U is written as split a $\langle \overline{o_p}, \overline{a_p} \rangle$, where $o_p : U \times I' \longrightarrow O'$ is the *observer*, attribute or output function, and $a_p : U \times I \longrightarrow \mathsf{B}(U \times O)$ stands for the coalgebra *action*, method or update function.

More specialised notions of a component will arise by instantiating T^{B}-interface parameters. Instantiations with $\mathbf{1}$ are interesting as they collapse part of the observation structure. For example, by making $I' = O' = \mathbf{1}$, one obtains $\mathsf{T}^{\mathsf{B}} \;=\; \mathsf{B}(\mathsf{Id} \times O)^{I}$, a shape for *functional components*. A possible classification follows.

- 'Functional' components:
$$\mathsf{T}^{\mathsf{B}}_{I,O} \;=\; \mathsf{B}(\mathsf{Id} \times O)^{I}$$
- 'Silent' components:
$$\mathsf{T}^{\mathsf{B}}_{I,O} \;=\; O^{I} \times \mathsf{B}$$
- 'Action' components:
$$\mathsf{T}^{\mathsf{B}}_{O} \;=\; \mathsf{B}(\mathsf{Id} \times O)$$
- 'Object' components:
$$\mathsf{T}^{\mathsf{B}}_{I,O} \;=\; O \times \mathsf{B}^{I}$$

Note that, by trivialising the behaviour monad, *i.e.*, making $\mathsf{B} = \mathsf{Id}$, the resulting 'functional' and 'object' components correspond, respectively, to the so-called *Mealy* and *Moore* machines in automaton theory. Recall that in a *Moore* automaton [Moo66] each state is associated to an output symbol, whereas such symbols are associated to transitions, rather than states, in a *Mealy* machine [Mea55].

'Silent' components, on the other hand, have structured attributes but their internal evolution is not triggered by the environment — at least by a no trivial stimulus. Finally, 'action' components also evolve with no external explicit trigger, but do produce an output in each evolution step. Taking $O = Act$, as a set of formal symbols standing for action names, and instantiating B with the (finite) powerset functor, the resulting (final) 'action' components would lead us back to CCS processes, as discussed earlier in this chapter.

We shall focus on the 'functional' and the 'object' shapes, since these capture the basic requirements we have made for a component model. However, they entail quite different interaction disciplines which justify a separate treatment. Therefore, the former will be dealt with in the next chapter and the latter in chapter 6.

64. BEHAVIOUR MONADS. As mentioned in the previous paragraph, the functor T^{B} is parametrized by a (strong) monad (§A.9), B, acting as a *behavioural model*. This means that the computation of a component action will not simply produce an output and a continuation state, but a B-structure of such pairs. The monadic structure provides the basic tools to handle such computations. Unit (η) and multiplication (μ), provide, respectively, a value embedding and a 'flatten' operation to reduce nested behavioural annotations. Strength, in either right (τ_r) or left (τ_l) versions, on the other hand, becomes crucial to handle context information. We have already resorted to a monadic structure in the study of process calculi in the previous sections. Such a structure becomes even more relevant now as all the component combinators, to be introduced along the next two chapters, are totally parametric on the monad, therefore abstracting away from any concrete behavioural model. It also turns out that monad commutativity (§A.10) is a welcome property, although not crucial.

Several possibilities can be considered, on a pragmatic basis, in the definition of B. The simplest case is, obviously, the *identity* monad, Id. Components, would then behave in a totally *deterministic* way. More interesting possibilities, capturing more complex behavioural features, include:

- *Partiality*, *i.e.*, the possibility of deadlock or failure, captured by the usual maybe monad, $\mathsf{B} = \mathsf{Id} + \mathbf{1}$ (§A.3).
- *Nondeterminism*, introduced by the (finite) powerset monad, $\mathsf{B} = \mathcal{P}$ (§A.3).
- *Ordered nondeterminism*, based on the (finite) sequence monad, $\mathsf{B} = \mathsf{Id}^*$.
- Monoidal *stamping*, with $\mathsf{B} = \mathsf{Id} \times M$. Notice that, for B to form a monad parameter M should support a monoidal structure to be used in the definition of η and μ (§A.5).
- *'Metric' nondeterminism* capturing situations in which, among the possible future evolutions of the component, some are more probable (cheaper, or more secure, or more ...) than others. In fact, for X finite, isomorphism $\mathcal{P}X \cong X \rightharpoonup \mathbf{1}$ (where $A \rightharpoonup B$ denotes the space of partial functions from A to B) suggests the extension of finite powerset to mappings, expressing a richer notion of nondeterminism in which each possible state is assigned a confidence level.

All of them are known to be strong monads in Set. The first two and the last one are commutative; the third is not. Commutativity of 'monoidal stamping' depends, of course, on commutativity of the underlying monoid. Moreover, they can safely be composed (§A.11).

65. THE BAG MONAD. The formal model for what we have called 'metric nondeterminism' is the *bag monad*. This is based on a structure $\langle M, \oplus, \otimes \rangle$, where both $\oplus$ and $\otimes$ define Abelian monoids over M and the latter distributes over the former. A bag of X is defined as a partial function space $X \rightharpoonup M$, modelled in the sequel as $\mathcal{P}(X \times M)$ subject to the following *functional dependence* invariant:

$$\mathsf{fdp} \;=\; \lambda m \,.\; \forall_{r,t \in m} \,.\; \pi_2\, r = \pi_2\, t \;\Rightarrow\; r = t$$

There are two basic operations on bags: *bag union*, denoted by $\uplus_M$, and *bag product*, denoted $\times_M$, defined by:

$$\begin{aligned} m_1 \uplus_M m_2 \;=\; & \{\langle \mathsf{ap}(m_1, x) \;\oplus\; \mathsf{ap}(m_2, x), x \rangle |\; x \in \mathsf{dom}\, m_1 \cap \mathsf{dom}\, m_2 \} \\ & \cup\; (m_1 \cup m_2 - (m_1 \cap m_2)) \end{aligned}$$

and

$$\times_M (m, e) \;=\; \mathcal{P}(\mathsf{id} \times \otimes_e)\, m \qquad \text{where } \otimes_e \; m = m \otimes e$$

Note that $\cup$ and $\cap$ are valid operators over bags as the underlying structure is a set. On the other hand, invariant fdp, makes meaningful the following operators borrowed from the algebra of mappings:

$$\mathsf{ap}(m, x) \;=\; \mathsf{the}\; (\mathcal{P}\pi_1)\; \{t \in m |\; \pi_2\, t = x\} \qquad \text{where } \mathsf{dom} \;=\; \mathcal{P}\pi_2$$

To characterise the bag monad we have to define the action of a functor $\mathsf{Bag}_M = \mathcal{P}(\mathsf{Id} \times M)_{\mathsf{fdp}}$ on functions and, of course, η and μ. Strength is inherited from the powerset monad. Thus, denoting by $\biguplus_M$ the reduction of $\uplus_M$,

$$\begin{aligned} \mathsf{Bag}_M\, f \;&=\; \biguplus_M \cdot \mathcal{P}(\mathsf{sing} \cdot (f \times \mathsf{id})) \\ \eta \;&=\; \mathsf{sing} \cdot \langle \mathsf{id}, \underline{1} \cdot ! \rangle \\ \mu \;&=\; \biguplus_M \cdot \mathcal{P} \times_M \end{aligned}$$

The following are two possible instantiations of this monad entailing two behaviour models that might be interesting to consider:

- *Cost* components: based on Bag_M for $M = \langle \mathbb{N}, +, \times \rangle$, which is just the usual notion of a bag or *multiset*. Components with such a behaviour model assign a cost to each alternative, which may be interpreted as, *e.g.*, a performance measure. Such 'costs' are added when components get composed.

This corresponds to the nondeterministic generalisation of *monoidal stamping* above.

- *Probabilistic* components: based on $M = \langle [0,1], \mathsf{min}, \times \rangle$ with the additional requirement that, for each $m \in \mathsf{Bag}_M$, $\sum(\mathcal{P}\pi_2)m = 1$. This assigns probabilities to each possible evolution of a component, introducing a (elementary) form of probabilistic nondeterminism.

66. BEHAVIOURAL CONVERSION. Sometimes there is a need to change the behavioural model of a particular component. This is typically the case when trying to compose two components originally defined under distinct behavioural assumptions. The condition under which such a conversion becomes possible is the existence of a monad morphism (§A.12) between the two models. Every such morphism $h : \mathsf{B} \Longrightarrow \mathsf{B}^{'}$ lifts to a natural transformation

$$\mathsf{T}^{h} : \mathsf{T}^{\mathsf{B}} \Longrightarrow \mathsf{T}^{\mathsf{B}^{'}}$$

transforming each T^{B} component into a $\mathsf{T}^{\mathsf{B}^{'}}$ one, over the same state space.

67. EXAMPLE. Not all behavioural conversions are possible, the rule being that the target monad should have 'enough' structure to embed the other. In any case, all that is left to verify is the existence of a monad morphism. As an example, we shall prove here that *partial* components can always be converted into *nondeterministic* ones.

Let $\mathsf{B} = \mathsf{Id} + \mathbf{1}$ and $\mathsf{B}' = \mathcal{P}$, as usual. We claim that $h = [\mathsf{sing}, \underline{\emptyset}]$ is a monad morphism.

Proof. Recall the definitions of these monads from §A.3. The two conditions stated on §A.12 have to be verified. Thus,

$$\begin{array}{cl}
 & h \cdot \eta^{\mathsf{B}} \\
= & \quad \{ \ \eta^{\mathsf{B}} \text{ definition } \} \\
 & [\mathsf{sing}, \underline{\emptyset}] \cdot \iota_1 \\
= & \quad \{ \ + \text{ cancellation } \} \\
 & \mathsf{sing} \\
= & \quad \{ \ \eta^{\mathsf{B}^{'}} \text{ definition } \} \\
 & \eta^{\mathsf{B}^{'}}
\end{array}$$

and

$$\begin{array}{cl}
 & h \cdot \mu^{\mathsf{B}} \\
= & \quad \{ \ \mu^{\mathsf{B}} \text{ definition } \}
\end{array}$$

$$
\begin{array}{cl}
 & [\mathsf{sing}, \underline{\emptyset}] \cdot [\mathsf{id}, \iota_2] \\
= & \quad \{ \ + \text{fusion} \ \} \\
 & [[\mathsf{sing}, \underline{\emptyset}] \cdot \mathsf{id}, [\mathsf{sing}, \underline{\emptyset}] \cdot \iota_2] \\
= & \quad \{ \ + \text{cancellation, identity} \ \} \\
 & [[\mathsf{sing}, \underline{\emptyset}], \underline{\emptyset}] \\
= & \quad \{ \ \text{set union} \ \} \\
 & [[\bigcup \cdot \mathsf{sing} \cdot \mathsf{sing}, \bigcup \cdot \mathsf{sing} \cdot \underline{\emptyset}], \bigcup \cdot \underline{\emptyset}] \\
= & \quad \{ \ + \text{fusion} \ \} \\
 & \bigcup \cdot [[\mathsf{sing} \cdot \mathsf{sing}, \mathsf{sing} \cdot \underline{\emptyset}], \underline{\emptyset}] \\
= & \quad \{ \ + \text{fusion, identity} \ \} \\
 & \bigcup \cdot [\mathsf{sing} \cdot [\mathsf{sing}, \underline{\emptyset}], \underline{\emptyset} \cdot \mathsf{id}] \\
= & \quad \{ \ + \text{absorption} \ \} \\
 & \bigcup \cdot [\mathsf{sing}, \underline{\emptyset}] \cdot ([\mathsf{sing}, \underline{\emptyset}] + \mathbf{1}) \\
= & \quad \{ \ \mu^{\mathsf{B}'} \text{ and } \mathsf{B} \text{ definitions} \ \} \\
 & \mu^{\mathsf{B}'} \cdot h \cdot \mathsf{B}\ h
\end{array}
$$

□

This extends to the following natural transformation between ‘functional’ components:

$$h^I : (\mathsf{Id} \times O + \mathbf{1})^I \Longrightarrow \mathcal{P}(\mathsf{Id} \times O)^I$$

Similarly, for ‘object’ components, yields

$$\mathsf{id}_O \times h^I : O \times (\mathsf{Id} + \mathbf{1})^I \Longrightarrow O \times \mathcal{P}\mathsf{Id}^I$$

Another example of a behavioural conversion is given by the following monad morphism from Bag_M for the probabilistic case above, to the finite powerset, suggested in [Man98]:

$$h_t \ = \ \mathcal{P}\pi_1 \cdot \mathsf{filter}_t$$

where $t \in [0, 1]$ is a threshold value and $\mathsf{filter}_t\ m = \{p \in m |\ \pi_2\ p > t\}$. Lifted to components it provides a ‘plain’ nondeterministic view of ‘probabilistic’ components, retaining only their ‘most probable’ behaviour.

68. This chapter introduced coalgebraic models for both *processes* and *components*. We have also shown how process calculi can be developed in a parametric and essentially pointfree way and their prototype implementations encoded in CHARITY. The next step is to develop similar calculi for software components. Two difficulties seem to arise. Firstly, components are inherently more complex than processes, as discussed in this last section. On the other hand, if there is no such thing as a 'canonical' process calculus, the characterisation of what a good notion of software component and component calculus is also remains an open question. The next two chapters intend to bring a preliminary contribution to this topic.

CHAPTER 5

Components as Arrows

Summary

This chapter introduces an algebra of components modelled as concrete seeded coalgebras for the class of Set *endofunctors* $\mathsf{T}^{\mathsf{B}} = \mathsf{B}(\mathsf{Id} \times O)^I$, *where* B *stands for a strong monad. It is shown how component interfaces and components themselves become, respectively, the objects and arrows of a bicategory* Cp_{B}. *Several combinators are defined on top of this structure and their properties investigated. A category of behaviours is derived from* Cp_{B} *as the appropriate universe to discuss components from a purely behavioural point of view. It is also shown how components' behaviour can be prototyped in* CHARITY.

1. A (bi)Category of Components

1. INTRODUCTION. This chapter introduces a calculus of software components modeled as seeded coalgebras for

$$\mathsf{T}^{\mathsf{B}} \;=\; \mathsf{B}(\mathsf{Id} \times O)^I \tag{5.1}$$

where B is a strong monad and I, O denote the types of input and output parameters, respectively. In §4.63 such components were referred to as *functional* to stress the explicit input-output correspondence. In fact, for $\mathsf{B} = \mathsf{Id}$, T^{B}-coalgebras over the singleton set $\mathbf{1}$ correspond exactly to functions from I to O. A relationship with what is called a Mealy machine [Mea55] in automata theory has already been mentioned in the previous chapter.

Components defined in this way organise themselves into a bicategory Cp_{B} whose *objects* are sets, standing for interface (or observation) universes, *arrows* are seeded T^{B}-coalgebras and *2-cells* are the correspondent comorphisms. Whenever clear from the context, Cp_{B} will be simply referred to as Cp. Formally,

2. Definition. The bicategory Cp_{B} is defined by

- A collection of sets, acting as component interfaces.
- For each pair $\langle I, O\rangle$ of objects, a hom-category $\mathsf{Cp}(I, O)$, whose objects are seeded $\mathsf{T}^{\mathsf{B}}_{I,O}$-coalgebras and arrows are seed preserving comorphisms. More specifically, a component p is specified as a pair $\langle u_p \in U_p, \overline{a}_p : U_p \longrightarrow \mathsf{B}(U_p \times O)^I\rangle$, where u_p is the seed value and the coalgebra dynamics is captured by currying a state-transition function $a_p : U_p \times I \longrightarrow \mathsf{B}\ (U_p \times O)$. An arrow $h : \langle u_p \in U_p, \overline{a}_p\rangle \longrightarrow \langle u_q \in U_q, \overline{a}_q\rangle$ satisfies the following *comorphism* and *seed preservation* conditions:

$$\overline{a}_q \cdot h \ = \ \mathsf{T}^{\mathsf{B}}\, h \cdot \overline{a}_p \tag{5.2}$$

$$h\ u_p \ = \ u_q \tag{5.3}$$

Composition is inherited from Set and the identity $1_p : p \longrightarrow p$, on component p, is defined as the identity id_{U_p} on the carrier of p[1].

- For each triple of objects $\langle I, K, O\rangle$, a composition law given by a functor

$$;_{I,K,O} : \mathsf{Cp}(I, K) \times \mathsf{Cp}(K, O) \longrightarrow \mathsf{Cp}(I, O)$$

whose action on objects and 2-cells is, respectively,

  - $p\ ;q \ = \ \langle\langle u_p, u_q\rangle \in U_p \times U_q, \overline{a}_{p;q}\rangle$ where $a_{p;q} : U_p \times U_q \times I \longrightarrow \mathsf{B}(U_p \times U_q \times O)$ is detailed as follows:

$$\begin{aligned} a_{p;q} \ = U_p \times U_q \times I &\xrightarrow{\ \mathsf{xr}\ } U_p \times I \times U_q \xrightarrow{a_p \times \mathsf{id}} \mathsf{B}(U_p \times K) \times U_q \\ &\xrightarrow{\ \tau_r\ } \mathsf{B}(U_p \times K \times U_q) \xrightarrow{\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})} \mathsf{B}(U_p \times (U_q \times K)) \\ &\xrightarrow{\mathsf{B}(\mathsf{id}\times a_q)} \mathsf{B}(U_p \times \mathsf{B}(U_q \times O)) \xrightarrow{\ \mathsf{B}\tau_l\ } \mathsf{BB}(U_p \times (U_q \times O)) \\ &\xrightarrow{\mathsf{BBa}^\circ} \mathsf{BB}(U_p \times U_q \times O) \xrightarrow{\ \mu\ } \mathsf{B}(U_p \times U_q \times O) \end{aligned}$$

  - $h\ ; k \ = \ h \times k$

- For each object K, an identity law given by a functor

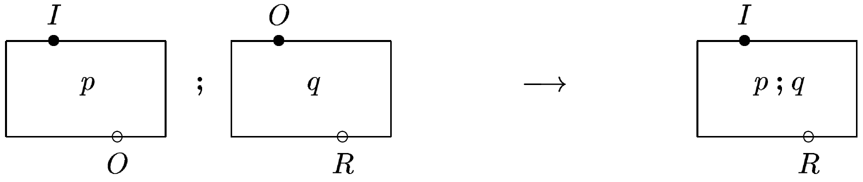
$$\mathsf{copy}_K : \mathbf{1} \longrightarrow \mathsf{Cp}(K, K)$$

whose action on objects is the constant component $\langle * \in \mathbf{1}, \overline{a}_{\mathsf{copy}_K}\rangle$, where $a_{\mathsf{copy}_K} \ = \ \eta_{(\mathbf{1}\times K)}$, which, by a slight abuse of notation, is also referred to

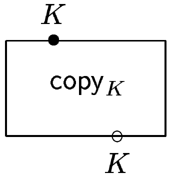
[1] For notational simplicity, we shall use a simple arrow ($\longrightarrow$) to declare any sort of morphism, including 2-cells, with the exception of natural transformations in Set. Should the notational convention of appendix B be strictly followed, 2-cells, such as the identity 1_p above, would be written as $1_p : p \Longrightarrow p$.

as copy_K. Similarly, the action on morphisms is the constant comorphism $\mathsf{id}_{\mathbf{1}}$.

3. Diagrams. Components are represented diagrammatically by rectangular boxes whose borders are marked with the respective input and output interface types. An input interface is marked by symbol $\bullet$, usually on the left of the box top line. The output type, on the other hand, follows a $\circ$ on the right of the box bottom line. Using this convention, the composition law and its identity are pictured as



and



Note that composition in Cp models *pipelining*. A pipe is formed by placing both components side by side and connecting the output of p to the input of q. Wherever p receives input, it executes and the output is fed to q. The monadic effect is propagated. For example, if p has the capacity of deadlocking and indeed deadlocks when receiving a particular stimulus, so does $p\,;q$.

4. Lemma. Let B be a strong monad. Then, Cp_{B} defined in §2 forms a bicategory.

Proof. Clearly each $\mathsf{Cp}(I,O)$ is a category, more exactly the category of $\mathsf{B}(\mathsf{Id}\times O)^{I}$-coalgebras over Set. It remains to be proved that both $;_{I,K,O}$ and copy_K are functors, for each I, K and O, as well as the existence of natural isomorphisms encoding ; associativity and unit. The reader is referred to appendix B, in particular to §B.2 and §B.6, for the definition of a bicategory.

The first part is almost trivial. For copy_K, let $\star$ be the object in the singleton category $\mathbf{1}$. Then $1_{\mathsf{copy}_K\star} = 1_{\langle *\in\mathbf{1},\overline{a}_{\mathsf{copy}_K}\rangle} = \mathsf{id}_{\mathbf{1}} = \mathsf{copy}_K\mathsf{id}_\star$ and preservation of composition holds trivially as copy_K is a constant functor. The two functoriality axioms are also easily verified for $;_{I,K,O}$. In fact, $1_p\,;1_q = \mathsf{id}_{U_p}\times\mathsf{id}_{U_q} = \mathsf{id}_{U_p\times U_q} = 1_{p;q}$, and, for h,h' (resp. k,k') $\mathsf{Cp}(I,K)$ (resp. $\mathsf{Cp}(K,O)$) -morphisms,

$$(h \cdot k) \mathbin{;} (h' \cdot k') = (h \cdot k) \times (h' \cdot k') = (h \times h') \cdot (k \times k') = (h \mathbin{;} h') \cdot (k \mathbin{;} k')$$

as expected. It has to be shown, however, that, given $h : p \longrightarrow p'$ and $k : q \longrightarrow q'$, $h \mathbin{;} k$ is actually a $\mathsf{Cp}(I, O)$-morphism from $p \mathbin{;} q$ to $p' \mathbin{;} q'$. Therefore, we have to show that $h \mathbin{;} k$ satisfies the comorphism condition

$$\mathsf{B}(h \times k)^I \cdot \overline{a}_{p;q} \;=\; \overline{a}_{p';q'} \cdot (h \times k)$$

Applying exponential absorption and fusion to the left hand and right hand side of this expression, respectively, we get $\overline{\mathsf{B}((h \times k) \times \mathsf{id}) \cdot a_{p;q}} = \overline{a_{p';q'} \cdot ((h \times k) \times \mathsf{id})}$. As *curry* is an isomorphism, it is enough to verify that

$$\mathsf{B}((h \times k) \times \mathsf{id}) \cdot a_{p;q} \;=\; a_{p';q'} \cdot ((h \times k) \times \mathsf{id})$$

In the sequel this transformation will be used in other similar situations. Thus,

$$\begin{aligned}
& a_{p';q'} \cdot ((h \times k) \times \mathsf{id}) \\
= \quad & \{ \; ; \text{ definition } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_{p'} \times \mathsf{id}) \cdot \mathsf{xr} \cdot ((h \times k) \times \mathsf{id}) \\
= \quad & \{ \; \mathsf{xr} \text{ natural, functors } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ((a_{p'} \cdot (h \times \mathsf{id})) \times k) \cdot \mathsf{xr} \\
= \quad & \{ \; \text{assumption: } \mathsf{B}(h \times \mathsf{id}) \cdot a_p = a_{p'} \cdot (h \times \mathsf{id}), \text{ functors } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{B}(h \times \mathsf{id}) \times k) \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \tau_r \text{ natural (C.5) } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}((h \times \mathsf{id}) \times k) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \mathsf{a}, \mathsf{xr} \text{ natural } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{B}(h \times (k \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \text{assumption: } \mathsf{B}(k \times \mathsf{id}) \cdot a_q = a_{q'} \cdot (k \times \mathsf{id}), \text{ functors } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(h \times \mathsf{B}(k \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \tau_l \text{ natural (C.6) } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{BB}(h \times (k \times \mathsf{id})) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \mathsf{a} \text{ natural } \} \\
& \mu \cdot \mathsf{BB}((h \times k) \times \mathsf{id}) \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \mu \text{ natural (C.16) } \} \\
& \mathsf{B}((h \times k) \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; ; \text{ definition } \}
\end{aligned}$$

$$\mathsf{B}((h \times k) \times \mathsf{id}) \cdot a_{p;q}$$

Finally notice that $h \mathbin{;} k$ preserves seeds because

$$(h \mathbin{;} k)u_{p;q} = (h \times k)\langle u_p, u_q \rangle = \langle hu_p, ku_q \rangle = \langle u_{p'}, u_{q'} \rangle = u_{p';q'}$$

We still have to show the existence of natural isomorphisms

$$\begin{aligned} \mathsf{a}_{I,K,L,O} &: \mathbin{;}_{I,K,O} \circ(\mathbin{;}_{K,L,O} \times \mathsf{Id}) \Longrightarrow \mathbin{;}_{I,L,O} \circ (\mathsf{Id} \times \mathbin{;}_{I,K,L}) \\ \mathsf{r}_{I,O} &: \mathbin{;}_{I,I,O} \circ(\mathsf{copy}_I \times \mathsf{Id}) \Longrightarrow \mathsf{Id} \\ \mathsf{l}_{I,O} &: \mathbin{;}_{I,O,O} \circ(\mathsf{Id} \times \mathsf{copy}_O) \Longrightarrow \mathsf{Id} \end{aligned}$$

whose components are invertible 2-cells

$$\begin{aligned} \mathsf{a}_{p,q,r} &: (p \mathbin{;} q) \mathbin{;} r \longrightarrow p \mathbin{;} (q \mathbin{;} r) \\ \mathsf{r}_p &: \mathsf{copy}_I \mathbin{;} p \longrightarrow p \\ \mathsf{l}_p &: p \mathbin{;} \mathsf{copy}_O \longrightarrow p \end{aligned}$$

for all components p, q and r of the appropriate types. Our task is simplified by the fact that each of these 2-cells is a component of a natural isomorphism in Set: respectively, associativity, right and left units. For example, $\mathsf{a}_{p,q,r}$ amounts to a_{U_p,U_q,U_r} in Set. As a consequence, the coherence conditions hold trivially. Therefore, we are left with proving that such arrows are in fact 2-cells in Cp_{B}. Clearly, in all three cases seeds are preserved. We check, now, the comorphism condition for the right unit, r. The proofs of the two remaining cases (concerning l and a) are deferred to [Appendix D, page 340].

Let $p : I \longrightarrow O$ be a component. The proof that r (more exactly, r_{U_p}) is a comorphism from $\mathsf{copy}_I \mathbin{;} p$ to p amounts to show the commutativity of the following diagram

$$\begin{tikzcd} (\mathbf{1} \times U_p) \arrow[rr, "\mathsf{r}"] \arrow[d, "\overline{a}_{\mathsf{copy}_I;p}"'] & & U_p \arrow[d, "\overline{a}_p"] \\ \mathsf{B}((\mathbf{1} \times U_p) \times O)^I \arrow[rr, "\mathsf{B}(\mathsf{r}\times\mathsf{id})^I"] & & \mathsf{B}(U_p \times O)^I \end{tikzcd}$$

i.e., $\mathsf{B}(\mathsf{r} \times \mathsf{id})^I \cdot \overline{a}_{\mathsf{copy}_I;p} = \overline{a}_p \cdot \mathsf{r}$, or, as noted above, $\mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot a_{\mathsf{copy}_I;p} = a_p \cdot (\mathsf{r} \times \mathsf{id})$. Thus

$$\begin{aligned} & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot a_{\mathsf{copy}_I;p} \\ = \quad & \{\ \mathbin{;} \text{ definition } \} \\ & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \times \mathsf{id}) \cdot \mathsf{xr} \\ = \quad & \{\ \eta \text{ strong (C.18) } \} \\ & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \eta \cdot \mathsf{xr} \\ = \quad & \{\ \mu \text{ and } \eta \text{ natural (C.16) and (C.17) } \} \\ & \mu \cdot \eta \cdot \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot \mathsf{xr} \\ = \quad & \{\ \text{monad associativity (C.15) and } \mathsf{xr} = \mathsf{xr}^\circ\ \} \end{aligned}$$

$$\begin{array}{cl}
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \\
= & \qquad \{ \text{ routine: } \mathsf{r} = (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{a}^\circ \ \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \\
= & \qquad \{ \ \tau_r \text{ unit (C.2) } \} \\
 & \mathsf{r} \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \\
= & \qquad \{ \ \mathsf{r} \text{ natural } \} \\
 & a_p \cdot \mathsf{r} \cdot \mathsf{a} \\
= & \qquad \{ \text{ routine: } \mathsf{r} \cdot \mathsf{a} = \mathsf{r} \times \mathsf{id} \ \} \\
 & a_p \cdot (\mathsf{r} \times \mathsf{id})
\end{array}$$

□

5. Remark. Recognising that components form a bicategory, amounts not only to define the first two combinators of the component's calculus, but also to set up its basic laws. Recall from §3.24 that the graph of a comorphism is a bisimulation. Therefore, the existence of a seed preserving comorphism between two components makes them T^{B}-bisimilar. That is exactly what we have done in the proof above, leading to the following laws, for appropriately typed components p, q and r:

$$\mathsf{copy}_I \,;p \ \sim \ p \ \sim \ p\,; \mathsf{copy}_O \tag{5.4}$$

$$(p\,;q)\,;r \ \sim \ p\,;(q\,;r) \tag{5.5}$$

6. Behaviour. The coalgebraic specification of a component describes immediate reactions to possible state/input configurations. Its extension in time becomes the component's *behaviour*. Formally, the behaviour $[\![(p)]\!]$ of a component p is computed by applying the induced *anamorphism* (§3.33) to the seed value of p. I.e.,

$$[\![(p)]\!] \ = \ [\![(\overline{a}_p)]\!] u_p$$

Behaviours organise themselves in a category Bh_{B}, or, simply, Bh, whose objects are sets and each arrow $b : I \longrightarrow O$ is an element of $\nu_{I,O}$, the carrier of the final coalgebra $\omega_{I,O}$ for functor $\mathsf{B}(\mathsf{Id} \times O)^I$. To define composition in Bh, first note that the definition of $\overline{a}_{p;q}$ in §2 actually introduces an operator — ; — between coalgebras: $\overline{a}_{p;q}$ could actually have been written as $\overline{a}_p\,;\overline{a}_q$. Therefore, we may define composition in Bh by a family of combinators, for each I, K and O,

$$\begin{aligned} ;_{I,K,O}{}^{\mathsf{Bh}} &: \mathsf{Bh}(I,K) \times \mathsf{Bh}(K,O) \longrightarrow \mathsf{Bh}(I,O) \\ ;_{I,K,O}{}^{\mathsf{Bh}} &= [\![(\omega_{I,K} \,;\, \omega_{K,O})]\!] \end{aligned}$$

On the other hand, identities are given by

$$\begin{aligned} \mathsf{copy}_K{}^{\mathsf{Bh}} &: \mathbf{1} \longrightarrow \mathsf{Bh}(K,K) \\ \mathsf{copy}_K{}^{\mathsf{Bh}} &= [\![(\overline{a}_{\mathsf{copy}_K})]\!] \; * \end{aligned}$$

i.e., the behaviour of component copy_K, for each K.

7. REMARK. It should be observed how the structure of Bh mirrors whatever structure Cp possesses. In fact, the former is isomorphic to a sub-(bi)category of the latter whose arrows are components defined over the corresponding final coalgebra. Alternatively, we may think of Bh as constructed by quotienting Cp by the greatest T^{B}-bisimulation. However, as final coalgebras are fully abstract with respect to bisimulation (§3.35), the bicategorical structure collapses: the hom-categories become simply hom-*sets*. Moreover, as discussed in the next section, some tensors in Cp_{B} become universal constructions in Bh, for some particular instances of B. That is also the reason why properties holding in Cp up to bisimulation, do hold 'on the nose' in the behaviours category. For example, we may rephrase laws (5.4) and (5.5), for suitably typed behaviours b, c and d, in Bh, as

$$\begin{aligned} \mathsf{copy}_I \,;\, b \;=\; b \;=\; b \,;\, \mathsf{copy}_O \\ (b \,;\, c) \,;\, d \;=\; b \,;\, (c \,;\, d) \end{aligned}$$

Prior to this, however, we have to check the lemma which follows.

8. LEMMA. Bh, as defined in §6, is a category and construction $[\![(\ )]\!]$ can be made into a 2-functor from Cp to Bh.

Proof. Let $b : I \longrightarrow O$ be a behaviour. Then,

$$\begin{aligned} & b \,;\, \mathsf{copy}_O \\ = & \quad \{ \text{ definition of } ; \text{ in } \mathsf{Bh} \} \\ & [\![(\omega_{I,O} \,;\, \mathsf{copy}_O)]\!] \langle b, * \rangle \\ = & \quad \{ \text{ law (5.4), ana definition } \} \\ & [\![(\omega_{I,O})]\!] b \\ = & \quad \{ \text{ ana reflection (3.8) } \} \end{aligned}$$

$$b$$

A similar calculation establishes $\mathsf{copy}_I \,;\, b = b$. On the other hand, for suitably typed behaviours b, c and d,

$$
\begin{array}{cl}
 & (b \,;\, c) \,;\, d \\
= & \qquad \{ \text{ definition of } ; \text{ in } \mathsf{Bh} \} \\
 & [\!((\omega_{I,K} \,;\, \omega_{K,L}) \,;\, \omega_{L,O})\!]\langle\langle b, c\rangle, d\rangle \\
= & \qquad \{ \text{ law (5.5), ana definition } \} \\
 & [\!(\omega_{I,K} \,;\, (\omega_{K,L} \,;\, \omega_{L,O}))\!]\langle b, \langle c, d\rangle\rangle \\
= & \qquad \{ \text{ definition of } ; \text{ in } \mathsf{Bh} \} \\
 & b \,;\, (c \,;\, d)
\end{array}
$$

So Bh is as a category. Now, to establish $[\!(\ )\!]$ as a 2-functor (§B.12 and 13), take the identity on objects as the object correspondence and, for each pair of objects $\langle I, O\rangle$ a functor $\mathsf{H}_{\,I,O} : \mathsf{Cp}(I, O) \longrightarrow \mathsf{Bh}(I, O)$ such that $\mathsf{H}_{\,I,O}p = [\!(\overline{a}_p)\!]u_p$ and $\mathsf{H}_{\,I,O}h = \mathsf{id}_{\nu_{I,O}}$, for every $\mathsf{Cp}(I, O)$ object p and arrow h. Note that each $\mathsf{Bh}(I, O)$ is simply a set (*i.e.*, a discrete category). Therefore, checking the functoriality conditions for $\mathsf{H}_{\,I,O}$ becomes trivial. Finally, because of the bicategorical structure collapse, the functorial laws hold as effective *equalities*, and not only up to a natural transformation, *i.e.*,

$$
\begin{array}{rcl}
[\!(\mathsf{copy}_K{}^{\mathsf{Cp}})\!] & = & \mathsf{copy}_K{}^{\mathsf{Bh}} \\
[\!((p \,;^{\mathsf{Cp}} q))\!] & = & [\!(p)\!] \;;^{\mathsf{Bh}} [\!(q)\!]
\end{array}
$$

The first equality is a direct consequence of the definition of $\mathsf{copy}_K{}^{\mathsf{Bh}}$. To establish the second one, recall, from the proof of §4, that, for composable components p and q, the product of two comorphisms with source in p and q, respectively, to, say, p' and q', is still a comorphism from the composite $p;q$ to $p';q'$. In particular, $\omega_{I,K} ; \omega_{K,O} \cdot ([\!(\overline{a}_p)\!] \times [\!(\overline{a}_q)\!]) = \mathsf{B}([\!(\overline{a}_p)\!] \times [\!(\overline{a}_q)\!]) \cdot \overline{a}_{p;q}$. Therefore, the fusion law for anamorphisms justifies the following calculation

$$
\begin{array}{cl}
 & [\!((p \,;^{\mathsf{Cp}} q))\!] \\
= & \qquad \{ \text{ } \mathsf{H} \text{ definition } \} \\
 & [\!(\overline{a}_{p;q})\!]\langle u_p, u_q\rangle \\
= & \qquad \{ \text{ ana fusion (3.9) } \} \\
 & [\!(\omega_{I,K} \,;\, \omega_{K,O})\!] \cdot ([\!(\overline{a}_p)\!] \times [\!(\overline{a}_q)\!])\langle u_p, u_q\rangle \\
= & \qquad \{ \text{ } ; \text{ definition in } \mathsf{Bh} \text{ } \} \\
 & ;^{\mathsf{Bh}} \cdot ([\!(\overline{a}_p)\!] \times [\!(\overline{a}_q)\!])\langle u_p, u_q\rangle \\
= & \qquad \{ \text{ function application } \} \\
 & ;^{\mathsf{Bh}}\langle [\!(\overline{a}_p)\!]u_p, [\!(\overline{a}_q)\!]u_q\rangle
\end{array}
$$

$$= \quad \{ \text{ H definition } \}$$

$$[\![p]\!] \; ;^{\mathsf{Bh}} [\![q]\!]$$

□

2. A Component Algebra

9. This section investigates the structure of Cp_{B} by introducing an algebra of T^{B}-components. The algebra is parametric on the behaviour model. Therefore, the definition of its combinators relies on generic properties of the strong monad B. It is shown that they are either lax endofunctors (§B.13) in Cp_{B} or simply functors between families of hom-categories. Their definition carries naturally to Bh_{B}, where they show up as behaviour connectives, defining a particular (typed) 'process' algebra.

Let us begin with the simple observation that functions can be regarded as a particular case of components, whose interfaces are given by their domain and codomain types. Formally,

10. REPRESENTATION OF FUNCTIONS. A function $f : A \longrightarrow B$ is represented in Cp as

$$\ulcorner f \urcorner \;=\; \langle * \in \mathbf{1}, \overline{a}_{\ulcorner f \urcorner} \rangle$$

i.e., as a coalgebra over $\mathbf{1}$ whose action is given by the currying of

$$a_{\ulcorner f \urcorner} = \mathbf{1} \times A \xrightarrow{\mathsf{id} \times f} \mathbf{1} \times B \xrightarrow{\eta_{(\mathbf{1} \times B)}} \mathsf{B}(\mathbf{1} \times B)$$

For example, taking B as the finite powerset monad, we get

$$\overline{a}_{\ulcorner f \urcorner} \; * \;=\; \lambda \, a \, . \, \{ \langle *, f \; a \rangle \}$$

In fact, any Set function lifts to Cp and keeps some, but not usually all, of its properties. Some cases are discussed in §§13, 14 and 15. First, however, we prove that, up to bisimulation, function lifting is functorial.

11. LEMMA. Let $g : I \longrightarrow K$ and $f : K \longrightarrow O$ be functions. Then,

$$\ulcorner f \cdot g \urcorner \sim \ulcorner g \urcorner \, ; \ulcorner f \urcorner \tag{5.6}$$

$$\ulcorner \mathsf{id}_I \urcorner \sim \mathsf{copy}_I \tag{5.7}$$

Proof. Equation (5.7) is an immediate consequence of the definition: it can actually be written as an equality. On the other hand, equation (5.6) is proved by showing that $\mathsf{r}_{\mathbf{1}} : \mathbf{1} \times \mathbf{1} \longrightarrow \mathbf{1}$ is a comorphism from $\ulcorner g \urcorner ; \ulcorner f \urcorner$ to $\ulcorner f \cdot g \urcorner$. As seeds are trivially preserved (because $\mathsf{r}\langle *, * \rangle = *$), r is also a Cp arrow and we are done. Therefore, all that remains to be proved is

$$\mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{\ulcorner g \urcorner}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_{\ulcorner f \urcorner} \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ definition $\}$

$$\mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \eta \cdot (\mathsf{id} \times g)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \cdot (\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ μ natural (C.16) $\}$

$$\mu \cdot \mathsf{BB}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \eta \cdot (\mathsf{id} \times g)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \cdot (\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ routine: $\mathsf{r} = (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{a}^\circ$ $\}$

$$\mu \cdot \mathsf{BBr} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \eta \cdot (\mathsf{id} \times g)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \cdot (\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ τ_l unit (C.2) $\}$

$$\mu \cdot \mathsf{Br} \cdot \mathsf{B}(\mathsf{id} \times \eta \cdot (\mathsf{id} \times g)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \cdot (\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ r natural $\}$

$$\mu \cdot \mathsf{B}\eta \cdot \mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{Br} \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \cdot (\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ monad unit (C.14) $\}$

$$\mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{Br} \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \cdot (\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ routine: $\mathsf{r} \cdot \mathsf{a} \cdot \mathsf{xr} = \mathsf{s} \cdot (\mathsf{r} \times \mathsf{id})$, functors $\}$

$$\mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{Bs} \cdot \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \tau_r \cdot (\eta \times \mathsf{id}) \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ η strong (C.18) $\}$

$$\mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{Bs} \cdot \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \eta \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ η natural (C.17) $\}$

$$\eta \cdot (\mathsf{id} \times g) \cdot \mathsf{s} \cdot (\mathsf{r} \times \mathsf{id}) \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ routine: $\mathsf{s} \cdot (\mathsf{r} \times \mathsf{id}) = (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{xr}$, xr natural $\}$

$$\eta \cdot (\mathsf{id} \times g) \cdot (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{xr} \cdot \mathsf{xr} \cdot (\mathsf{id} \times f)$$

$= \quad \{$ $\mathsf{xr} = \mathsf{xr}^\circ$ $\}$

$$\eta \cdot (\mathsf{id} \times g) \cdot (\mathsf{r} \times \mathsf{id}) \cdot (\mathsf{id} \times f)$$

$= \quad \{$ $\times$ functor $\}$

$$\eta \cdot (\mathsf{id} \times (g \cdot f)) \cdot (\mathsf{r} \times \mathsf{id})$$

$= \quad \{$ definition $\}$

$$a_{\ulcorner g \cdot f \urcorner} \cdot (\mathsf{r} \times \mathsf{id})$$

□

12. REMARK. Note the irrelevance of the state space in any component intended to represent a function. In fact, consider an alternative representation of f in Cp as $f' = \langle u \in U, \overline{\eta_{U\times B} \cdot (\mathsf{id} \times f)} \rangle$, for an arbitrary set U and $u \in U$. Then, $f' \sim \ulcorner f \urcorner$, witnessed by $!_U : U \longrightarrow \mathbf{1}$. Clearly, $!_U$ preserves seeds and η naturality entails

$$\eta_{\mathbf{1}\times B} \cdot (!_U \times f) \;=\; \mathsf{B}(!_U \times \mathsf{id}) \cdot \eta_{U\times B} \cdot (\mathsf{id} \times f)$$

which establishes $!_U$ as a comorphism from f' to $\ulcorner f \urcorner$.

13. LEMMA. Isomorphisms, split monos and split epis lift to Cp as, respectively, isomorphisms, split monos and split epis.

Proof. Let $f : A \longrightarrow B$ be a Set-isomorphism. Then

$$\begin{array}{cl}
 & \ulcorner f \urcorner \,;\, \ulcorner f^{\circ} \urcorner \\
\sim & \quad \{ \text{ law (5.6) } \} \\
 & \ulcorner f^{\circ} \cdot f \urcorner \\
\sim & \quad \{ \text{ } f \text{ isomorphism } \} \\
 & \ulcorner \mathsf{id}_A \urcorner \\
\sim & \quad \{ \text{ law (5.7) } \} \\
 & \mathsf{copy}_A
\end{array}$$

Conversely, $\ulcorner f^{\circ} \urcorner ; \ulcorner f \urcorner \sim \ulcorner f \cdot f^{\circ} \urcorner \sim \ulcorner \mathsf{id}_B \urcorner \sim \mathsf{copy}_B$. For f a split epi (respectively, a split mono) consider the first (respectively, second) part of the proof above, taking f° as a section (respectively, a retraction) of f. This result is broadly applicable as every epi and every mono with non empty source split in Set.

□

14. INITIAL OBJECT. Lifting canonical Set arrows to Cp is a simple way to explore the structure of Cp itself. For instance, consider the lifting of $?_I : \emptyset \longrightarrow I$ (§2.6). Clearly, $?_I$ keeps its naturality as, for any $p : I \longrightarrow O$, the following diagram commutes up to bisimulation,

$$\begin{tikzcd}
I \arrow[r, "p"] & O \\
\emptyset \arrow[u, "\ulcorner ?_I \urcorner"] \arrow[ur, "\ulcorner ?_O \urcorner"'] &
\end{tikzcd}$$

because both $\ulcorner ?_I \urcorner$ and $\ulcorner ?_O \urcorner$ are the *inert* components: the absence of input makes reaction impossible. More formally, let us prove that

$$\ulcorner ?_I \urcorner \,;\, p \;\sim\; \ulcorner ?_O \urcorner \tag{5.8}$$

holds.

Proof. We will show that $!_{\mathbf{1}\times U_p} : \mathbf{1} \times U_p \longrightarrow \mathbf{1}$ is an arrow in Cp from $\ulcorner ?_I \urcorner ; p$ to $?_O$. Seeds being trivially preserved (because $!\langle *, u_p\rangle = * = u_{\ulcorner ?_O \urcorner}$), what remains to be shown is the comorphism condition, *i.e.*, the commutativity of the following diagram:

$$\begin{tikzcd}
\mathbf{1} \times U_p \arrow[r, "!"] \arrow[d, "\overline{a}_{\ulcorner ?_I \urcorner ; p}"'] & \mathbf{1} \arrow[d, "\overline{a}_{\ulcorner ?_O \urcorner}"] \\
\mathsf{B}((\mathbf{1} \times U_p) \times O)^{\emptyset} \arrow[r, "\mathsf{B}(! \times \mathsf{id})^{\emptyset}"] & \mathsf{B}(\mathbf{1} \times O)^{\emptyset}
\end{tikzcd}$$

Thus,

$$\begin{array}{cl}
 & \mathsf{B}(! \times \mathsf{id})^{\emptyset} \cdot \overline{a}_{\ulcorner ?_I \urcorner ; p} \\
= & \qquad \{\ f = ?_B \cdot \mathsf{zr}_A \text{ for any function } f : A \times \emptyset \longrightarrow B\ \} \\
 & \mathsf{B}(! \times \mathsf{id})^{\emptyset} \cdot \overline{?_{\mathsf{B}((\mathbf{1}\times U_p)\times O)} \cdot \mathsf{zr}_{(\mathbf{1}\times U_p)}} \\
= & \qquad \{\ \text{exponential absorption (2.40)}\ \} \\
 & \overline{\mathsf{B}(! \times \mathsf{id}) \cdot ?_{\mathsf{B}((\mathbf{1}\times U_p)\times O)} \cdot \mathsf{zr}_{(\mathbf{1}\times U_p)}} \\
= & \qquad \{\ \text{initiality (2.5)}\ \} \\
 & \overline{?_{\mathsf{B}(\mathbf{1}\times O)} \cdot \mathsf{zr}_{\mathbf{1}\times U_p}} \\
= & \qquad \{\ \mathsf{zr} \text{ natural}\ \} \\
 & \overline{?_{\mathsf{B}(\mathbf{1}\times O)} \cdot \mathsf{zr}_{\mathbf{1}} \cdot (!_{(\mathbf{1}\times U_p)} \times \mathsf{id})} \\
= & \qquad \{\ f = ?_B \cdot \mathsf{zr}_A \text{ for any function } f : A \times \emptyset \longrightarrow B \text{ and } \ulcorner f \urcorner \text{ definition}\ \} \\
 & \overline{a_{\ulcorner ?_O \urcorner} \cdot (!_{(\mathbf{1}\times U_p)} \times \mathsf{id})} \\
= & \qquad \{\ \text{exponential fusion (2.39)}\ \} \\
 & \overline{a}_{\ulcorner ?_O \urcorner} \cdot !_{\mathbf{1}\times U_p}
\end{array}$$

□

Equation (5.8) lifts to an equality in Bh, as does any other bisimulation equation in Cp, as discussed in §6. Therefore, $\emptyset$ is the *initial* object in Bh.

15. LIFTING !. A different situation occurs when lifting $!_I : I \longrightarrow \mathbf{1}$ because naturality is lost. In fact, the following diagram fails to commute for non trivial B

$$\begin{tikzcd}
I \arrow[r, "p"] \arrow[d, "\ulcorner !_I \urcorner"'] & O \arrow[dl, "\ulcorner !_O \urcorner"] \\
\mathbf{1}
\end{tikzcd}$$

To check this, take B as the finite powerset monad. Clearly, $p \,;\, \ulcorner !_O \urcorner$ will deadlock whenever p does. By 'deadlocking' we mean the empty set of responses is produced. On the other hand, $\ulcorner !_I \urcorner$ never deadlocks as this is prevented by the definition of function lifting in §10. Therefore, the two components are not bisimilar and so $\mathbf{1}$ does not become the final object in Bh_{B}, for non trivial monads. It is, however, the final object in the behaviours category of deterministic components (*i.e.*, for $\mathsf{B} = \mathsf{Id}$).

16. WIRES. Components over $\mathbf{1}$ which are defined from identities using only the structural properties of the underlying category — *i.e.*, arrows associated to products, coproducts and exponentials — are called *wires*. Their role is essentially to provide 'interconnection buses' between other components. As it will become evident soon, the fact that Cp has interface types as objects, often requires wiring in order to perform a legal connection.

Typical examples of wires are the liftings of some canonical isomorphisms — like a, s, l or r. Connecting components through isomorphisms leads to a particularly simple notion of *interchangeablity* (to be discussed later in §7.18): bisimilarity up to an isomorphic rearranging of the interface (referred to as 'up to isomorphic wiring' in the sequel). Clearly, however, some components are bisimilar only up to non isomorphic wiring. Examples of this last sort of connectors are the liftings of embeddings, projections, and, in particular, *codiagonals* and *diagonals* given by

$$\begin{aligned}
\nabla : I + I \longrightarrow I &= [\mathsf{id}_I, \mathsf{id}_I] \\
\vartriangle : O \longrightarrow O \times O &= \langle \mathsf{id}_O, \mathsf{id}_O \rangle
\end{aligned}$$

used to *merge* input and *replicate* output types. For example, a component $p : I \longrightarrow O$ can be prepared to accept input from different sources and, simultaneously, to deliver output to distinct targets by performing the composition

$$\ulcorner \nabla \urcorner \,;\, p \,;\, \ulcorner \vartriangle \urcorner$$

which is typed as $I + I \longrightarrow O \times O$

17. WRAPPING. The pre- and post-composition of a component with Cp-lifted functions, can be encapsulated in an unique combinator, called *wrapping*, which may be thought of as an extension of the *renaming* connective found in process calculi (as seen in chapter 4). Let $p : I \longrightarrow O$ be a component and consider functions $f : I' \longrightarrow I$ and $g : O \longrightarrow O'$. Component p wrapped by f and g is denoted by $p[f, g]$ and has type $I' \longrightarrow O'$. It is defined by input pre-composition with f and output post-composition with g. Formally, the wrapping combinator is a functor

$$-[f, g] : \mathsf{Cp}(I, O) \longrightarrow \mathsf{Cp}(I', O')$$

which is the identity on morphisms and maps a component $\langle u_p \in U_p, \overline{a}_p \rangle$ into $\langle u_p \in U_p, \overline{a}_{p[f,g]} \rangle$, where

$$a_{p[f,g]} = U_p \times I' \xrightarrow{\mathsf{id} \times f} U_p \times I \xrightarrow{a_p} \mathsf{B}(U_p \times O) \xrightarrow{\mathsf{B}(\mathsf{id} \times g)} \mathsf{B}(U_p \times O')$$

18. LEMMA. For any functions $f : I' \longrightarrow I$ and $g : O \longrightarrow O'$, the *wrapping* combinator $-[f, g]$ is a functor from $\mathsf{Cp}(I, O)$ to $\mathsf{Cp}(I', O')$.

Proof. Having defined $-[f, g]$ as the identity on morphisms, the functoriality conditions hold trivially once proved that any $\mathsf{Cp}(I, O)$-arrow $h : p \longrightarrow q$ is also a $\mathsf{Cp}(I', O')$-arrow from $p[f, g]$ to $q[f, g]$. Therefore,

$$\begin{array}{cl}
 & a_{q[f,g]} \cdot (h \times \mathsf{id}) \\
= & \quad \{ \text{ wrapping definition } \} \\
 & \mathsf{B}(\mathsf{id} \times g) \cdot a_q \cdot (\mathsf{id} \times f) \cdot (h \times \mathsf{id}) \\
= & \quad \{ \text{ assumption: } h : p \longrightarrow q \ \} \\
 & \mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_p \cdot (\mathsf{id} \times f) \\
= & \quad \{ \ \mathsf{B} \text{ functor, identity } \} \\
 & \mathsf{B}(h \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times g) \cdot a_p \cdot (\mathsf{id} \times f) \\
= & \quad \{ \text{ wrapping definition } \} \\
 & \mathsf{B}(h \times \mathsf{id}) \cdot a_{p[f,g]}
\end{array}$$

□

19. LEMMA. For any component $p : I \longrightarrow O$ and functions $f : I' \longrightarrow I$, $f' : J \longrightarrow I'$, $g : O \longrightarrow O'$ and $g' : O' \longrightarrow R$,

$$(p[f, g])[f', g'] \sim p[f \cdot f', g' \cdot g] \tag{5.9}$$

Proof.

$$\begin{array}{ll}
& a_{(p[f,g])[f',g']} \\
= & \quad \{ \text{ wrapping definition } \} \\
& \mathsf{B}(\mathsf{id} \times g') \cdot a_{p[f,g]} \cdot (\mathsf{id} \times f') \\
= & \quad \{ \text{ wrapping definition } \} \\
& \mathsf{B}(\mathsf{id} \times g') \cdot \mathsf{B}(\mathsf{id} \times g) \cdot a_p \cdot (\mathsf{id} \times f) \cdot (\mathsf{id} \times f') \\
= & \quad \{ \text{ functors } \} \\
& \mathsf{B}(\mathsf{id} \times (g' \cdot g)) \cdot a_p \cdot (\mathsf{id} \times (f \cdot f')) \\
= & \quad \{ \text{ wrapping definition } \} \\
& a_{p[f \cdot f', g' \cdot g]}
\end{array}$$

□

20. LEMMA. For any component $p : I \longrightarrow O$ and functions $f : I' \longrightarrow I$ and $g : O \longrightarrow O'$,

$$p[f,g] \sim \ulcorner f \urcorner \,;\, p \,;\, \ulcorner g \urcorner \tag{5.10}$$

Proof. [Appendix D, page 344].

□

21. SPECIAL COMPONENTS. Some simple components arise by lifting elementary functions to Cp. We have already remarked in §14 that the lifting of the canonical arrow associated to the initial Set object plays the role of an *inert* component, unable to react to the outside world. It seems convenient to give this component a name:

$$\mathsf{inert}_A = \ulcorner ?_A \urcorner \tag{5.11}$$

In particular, we define the nil component as

$$\mathsf{nil} = \mathsf{inert}_\emptyset = \ulcorner ?_\emptyset \urcorner = \ulcorner \mathsf{id}_\emptyset \urcorner \tag{5.12}$$

typed as $\mathsf{nil} : \emptyset \longrightarrow \emptyset$. Note that any component $p : I \longrightarrow O$ can be made inert by wrapping. For example,

$$p[?_I, !_O] \sim \mathsf{inert}_{\mathbf{1}} \tag{5.13}$$

Proof.

$$
\begin{array}{ll}
 & p[?_I, !_O] \\
\sim & \quad \{ \text{ law (5.10) } \} \\
 & \ulcorner ?_I \urcorner \,;\, p \,;\, \ulcorner !_O \urcorner \\
\sim & \quad \{ \text{ law (5.8) } \} \\
 & \ulcorner ?_O \urcorner \,;\, \ulcorner !_O \urcorner \\
\sim & \quad \{ \text{ law (5.6) } \} \\
 & \ulcorner !_O \cdot ?_O \urcorner \\
\sim & \quad \{ \text{ initiality (2.5) } \} \\
 & \ulcorner ?_{\mathbf{1}} \urcorner \\
= & \quad \{ \text{ definition } \} \\
 & \mathsf{inert}_{\mathbf{1}}
\end{array}
$$

□

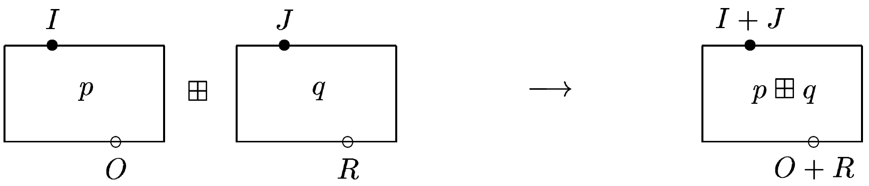
A somewhat dual role is played by component

$$\mathsf{idle} \;=\; \ulcorner \mathsf{id}_{\mathbf{1}} \urcorner \tag{5.14}$$

Notice that $\mathsf{idle} : \mathbf{1} \longrightarrow \mathbf{1}$ is always willing to propagate an unstructured stimulus (*e.g.*, the push of a button) leading to a (similarly) unstructured reaction (*e.g.*, exciting a led).

22. EXTERNAL CHOICE. Components can be aggregated in a number of ways different ways, besides the 'pipeline' composition already discussed in §2. In the following paragraphs we shall consider three such combinators and characterise them as lax functors in Cp.

The first composition pattern to be considered is *external choice*. Let $p : I \longrightarrow O$ and $q : J \longrightarrow R$ be two components defined by $\langle u_p \in U_p, \overline{a}_p \rangle$ and $\langle u_q \in U_q, \overline{a}_q \rangle$, respectively. When interacting with the composed system, environment will be allowed to choose either to input a value of type I or one of type J, which will trigger the corresponding component (p or q, respectively), producing the associated output. This is pictured as



A formal definition follows.

23. Definition. External choice is defined as a lax functor $\boxplus : \mathsf{Cp} \times \mathsf{Cp} \longrightarrow \mathsf{Cp}$, which consists of

- An action on objects given by
$$I \boxplus J \; = \; I + J$$
- A family of functors
$$\boxplus_{I,O,J,R} : \mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R) \longrightarrow \mathsf{Cp}(I+J, O+R)$$
yielding
$$p \boxplus q \; = \; \langle \langle u_p, u_q \rangle \in U_p \times U_q, \overline{a}_{p \boxplus q} \rangle$$
where

$$\begin{aligned}
a_{p\boxplus q} \; = \quad & U_p \times U_q \times (I+J) \xrightarrow{\mathsf{dr}} U_p \times U_q \times I + U_p \times U_q \times J \\
& \xrightarrow{\mathsf{xr}+\mathsf{a}} U_p \times I \times U_q + U_p \times (U_q \times J) \\
& \xrightarrow{a_p \times \mathsf{id} + \mathsf{id} \times a_q} \mathsf{B}\ (U_p \times O) \times U_q + U_p \times \mathsf{B}\ (U_q \times R) \\
& \xrightarrow{\tau_r + \tau_l} \mathsf{B}\ (U_p \times O \times U_q) + \mathsf{B}\ (U_p \times (U_q \times R)) \\
& \xrightarrow{\mathsf{Bxr}+\mathsf{Ba}^\circ} \mathsf{B}\ (U_p \times U_q \times O) + \mathsf{B}\ (U_p \times U_q \times R) \\
& \xrightarrow{[\mathsf{B}\ (\mathsf{id}\times\iota_1), \mathsf{B}\ (\mathsf{id}\times\iota_2)]} \mathsf{B}\ (U_p \times U_q \times (O+R))
\end{aligned}$$

and which maps pairs of arrows $\langle h_1, h_2 \rangle$ into $h_1 \times h_2$.

24. Lemma. Combinator $\boxplus$ is a lax functor in Cp. In particular, for any components p, q, p' and q',

$$(p \boxplus p') \; ; (q \boxplus q') \; \sim \; (p\; ; q) \boxplus (p' \; ; q') \tag{5.15}$$

$$\mathsf{copy}_{K \boxplus K'} \; \sim \; \mathsf{copy}_K \boxplus \mathsf{copy}_{K'} \tag{5.16}$$

Proof. The following steps have to be undertaken:

- For any sets I, O, J and R, $\boxplus_{I,O,J,R}$ is a functor from $\mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R)$ to $\mathsf{Cp}(I+J, O+R)$. Let $h : p \longrightarrow p'$ and $k : q \longrightarrow q'$ be component morphisms relating $p, p' : I \longrightarrow O$ and $q, q' : J \longrightarrow R$, respectively. Having defined $h \boxplus k$ as $h \times k$, the functoriality conditions arise directly from product functoriality. In fact, $(h' \boxplus h) \cdot (k' \boxplus k) = (h' \times h) \cdot (k' \times k) = (h' \cdot k') \times (h \cdot k) = (h' \cdot k') \boxplus (h \cdot k)$. Similarly, $1_p \boxplus 1_q = 1_p \times 1_q = \mathsf{id}_{U_p} \times \mathsf{id}_{U_q} = 1_{U_p \times U_q} = 1_{p \boxplus q}$. One has, however, to check that $h \times k$ is a Cp morphism from $p \boxplus q$ to $p' \boxplus q'$. This is proved in [Appendix D, page 346].

- For each tuple $\langle I, I', K, K', O, O' \rangle$ of objects, there exists a natural transformation

$$\mathsf{m}_{\boxplus} : ;_{I+I',K+K',O+O'} \circ (\boxplus_{I,K,I',K'} \times \boxplus_{K,O,K',O'}) \Longrightarrow \boxplus_{I,O,I',O'} \circ (;_{I,K,O} \times ;_{I',K',O'})$$

  whose component $\mathsf{m}_{p,q,p',q'}$, for each $p : I \longrightarrow K$, $q : K \longrightarrow O$, $p' : I' \longrightarrow K'$ and $q' : K' \longrightarrow O'$ is a Cp 2-cell, *i.e.*, a seed preserving comorphism from $(p \boxplus q) \,;(p' \boxplus q')$ to $(p \,; p') \boxplus (q \,; q')$. Let us take each $\mathsf{m}_{p,q,p',q'}$ as the component $\mathsf{m}_{U_p,U_q,U_{p'},U_{q'}} : (U_p \times U_q) \times (U_{p'} \times U_{q'}) \longrightarrow (U_p \times U_{p'}) \times (U_q \times U_{q'})$ of the Set natural isomorphism m. Then, naturality comes for free and seed preservation is trivially verified. The comorphism condition, however, has to be checked (see [Appendix D, page 346]). Also note that coherence is easily verified since the action of both $;$ and $\boxplus$ on 2-cells is function product. Therefore, $(\mathsf{a} \boxplus \mathsf{a}) \cdot \mathsf{m} \cdot (\mathsf{id} \,; \mathsf{m}) = (\mathsf{a} \times \mathsf{a}) \cdot \mathsf{m} \cdot (\mathsf{id} \times \mathsf{m}) = \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{a} = \mathsf{m} \cdot (\mathsf{m} \,; \mathsf{id}) \cdot \mathsf{a}$.
- Finally, for each pair $\langle K, K' \rangle$ of objects, there exists a natural transformation

$$\mathsf{u}_{\boxplus} : \mathsf{copy}_{K \boxplus K'} \Longrightarrow \boxplus_{K,K',K,K'} \circ (\mathsf{copy}_K \times \mathsf{copy}_{K'})$$

  i.e., a Cp 2-cell from $\mathsf{copy}_{K \boxplus K'}$ to $\mathsf{copy}_K \boxplus \mathsf{copy}_{K'}$. $\mathsf{u}_{\boxplus}$ is then a seed preserving comorphism connecting carriers $U_{\mathsf{copy}_{K \boxplus K'}} = \mathbf{1}$ to $U_{\mathsf{copy}_K \boxplus \mathsf{copy}_{K'}} = \mathbf{1} \times \mathbf{1}$. By taking $\mathsf{u}_{\boxplus}$ as $\mathsf{r}_{\mathbf{1}}^{\circ} : \mathbf{1} \longrightarrow \mathbf{1} \times \mathbf{1}$, naturality, seed preservation and coherence are immediately established. That $\mathsf{r}_{\mathbf{1}}^{\circ}$ satisfies the comorphism condition is shown in [Appendix D, page 346].

□

25. LEMMA. Let f and g be functions. Then

$$\ulcorner f \urcorner \boxplus \ulcorner g \urcorner \sim \ulcorner f + g \urcorner \tag{5.17}$$

Proof. [Appendix D, page 356].

□

26. LEMMA. Up to isomorphic wiring, $\boxplus$ is a symmetric tensor product in each hom-category, with nil as unit, *i.e.*,

$$(p \boxplus q) \boxplus r \sim (p \boxplus (q \boxplus r))[\mathsf{a}_+, \mathsf{a}_+{}^{\circ}] \tag{5.18}$$

$$\mathsf{nil} \boxplus p \sim p[\mathsf{r}_+, \mathsf{r}_+{}^{\circ}] \tag{5.19}$$

$$p \boxplus \mathsf{nil} \sim p[\mathsf{l}_+, \mathsf{l}_+{}^{\circ}] \tag{5.20}$$

$$p \boxplus q \sim (q \boxplus p)[\mathsf{s}_+, \mathsf{s}_+] \tag{5.21}$$

Proof. In general, to prove a bisimulation equation $p \sim q$, a function $h : U_p \longrightarrow U_q$ has to be identified and shown to form a seed preserving comorphism. Therefore, equations (5.18), (5.19), (5.20) and (5.21) are proved by establishing as Cp 2-cells the Set natural isomorphisms a, r, l and s, respectively. Seed preservation is obvious in all cases. The verification of the comorphism conditions can be found in [Appendix D, page 357]. Notice that (5.20) can be obtained from (5.19), and the other way round, using $\boxplus$ commutativity (5.21). In appendix D, however, we give a direct proof.

□

27. REMARK. Laws (5.18) to (5.21) can be alternatively stated as the observation that the canonical Set isomorphisms a_+, r_+, l_+ and s_+, once lifted to Cp, keep their naturality up to bisimulation. Consider, for example, the case of r_+. Naturality preservation means that

$$\begin{array}{ll} & (\mathsf{nil} \boxplus p) \,;\, \ulcorner \mathsf{r}_+ \urcorner \sim \ulcorner \mathsf{r}_+ \urcorner \,;\, p \\ \equiv & \qquad \{\ \text{Leibniz}\ \} \\ & (\mathsf{nil} \boxplus p) \,;\, \ulcorner \mathsf{r}_+ \urcorner \,;\, \ulcorner \mathsf{r}_+{}^\circ \urcorner \sim \ulcorner \mathsf{r}_+ \urcorner \,;\, p \,;\, \ulcorner \mathsf{r}_+{}^\circ \urcorner \\ \equiv & \qquad \{\ \mathsf{r}_+ \text{ isomorphism and lemma §13}\ \} \\ & (\mathsf{nil} \boxplus p) \sim \ulcorner \mathsf{r}_+ \urcorner \,;\, p \,;\, \ulcorner \mathsf{r}_+{}^\circ \urcorner \end{array}$$

which, by lemma §20, equivales to equation (5.19).

As a general remark, note that component laws admit different formulations depending on wiring. For example, equations (5.19) and (5.20) may also be presented as

$$(\mathsf{nil} \boxplus p)[\mathsf{r}_+{}^\circ, \mathsf{r}_+] \sim p \sim (p \boxplus \mathsf{nil})[\mathsf{l}_+{}^\circ, \mathsf{l}_+]$$

Finally observe that in the lemma above, and wherever the expression 'up to isomorphic wiring' is used, the bisimulation involved is witnessed by a 2-cell isomorphism.

28. AN EITHER CONSTRUCTION. After studying the choice combinator in some detail, the question arises as whether there is a counterpart in Cp to the *either* construction in Set. The answer is only partly positive. Let $p : I \longrightarrow O$ and $q : J \longrightarrow O$ be two components sharing a common output type O, and define

$$[p, q] = (p \boxplus q) \,;\, \ulcorner \nabla \urcorner$$

Lemma §29 below shows that the following diagram commutes up to bisimulation:

$$\begin{tikzcd}
I \arrow[r, "\ulcorner \iota_1 \urcorner"] \arrow[dr, "p"'] & I \boxplus J \arrow[d, "{[p,q]}"] & J \arrow[l, "\ulcorner \iota_2 \urcorner"'] \arrow[dl, "q"] \\
& O &
\end{tikzcd}$$

i.e.,

$$\ulcorner \iota_1 \urcorner \,;\, [p, q] \sim p \tag{5.22}$$

$$\ulcorner \iota_2 \urcorner \,;\, [p, q] \sim q \tag{5.23}$$

However, $[p, q]$ is not the unique arrow making the diagram to commute. Therefore,

29. LEMMA. The choice combinator, $\boxplus$, lifts to a *weak coproduct* in Bh.

Proof. A weak coproduct is defined like a coproduct but for the uniqueness of the mediating arrow (the *either* construction). Existence, *i.e.*, the validity of (5.22) and (5.23), is proved in [Appendix D, page 365]. The lemma follows by lifting the diagram to Bh.

What we would like to stress here is the impossibility of turning *either* into an universal construction in Bh. The basic observation is that codiagonal $\triangledown$ does not keep its naturality when lifted to Cp. In fact, a counterexample can be found even in the simple setting of deterministic components (*i.e.*, with $\mathsf{B} = \mathsf{Id}$). Let $p = \langle 0 \in \mathbb{N}, \overline{a}_p \rangle : \mathbb{N} \longrightarrow \mathbb{N}$ be such that, upon receiving an input i, i is added to the current state value and the result sent to the output. Consider the following sequence of inputs (of type $\mathbb{N} + \mathbb{N}$): $s = \langle \iota_1 5, \iota_2 3, \iota_1 4, ... \rangle$. The reaction to s of $\ulcorner \triangledown \urcorner \,;\, (p \boxplus p)$ is $\langle 5, 3, 9, \ldots \rangle$ while $p \,;\, \ulcorner \triangledown \urcorner$, resorting only to one copy of p, produces $\langle 5, 8, 12, \ldots \rangle$.

□

30. Failing universality means there is not a *fusion* law for $\boxplus$, even in the deterministic case. However, *cancellation*, *reflection* and *absorption* laws do hold strictly in Bh and, up to bisimulation, in Cp. Cancellation has just been proved in §29. The other two are considered in the following lemma.

31. LEMMA. For suitably typed components p, q, p' and q',

$$[\ulcorner \iota_1 \urcorner, \ulcorner \iota_2 \urcorner] \sim \mathsf{copy}_{I+J} \tag{5.24}$$

$$(p \boxplus q) \,;\, [p', q'] \sim [p \,;\, p', q \,;\, q'] \tag{5.25}$$

Proof.

$$
\begin{array}{ll}
& [\ulcorner \iota_1 \urcorner, \ulcorner \iota_2 \urcorner] \\
\sim & \quad \{\ \text{definition of } \textit{either} \text{ in } \mathsf{Cp}\ \} \\
& (\ulcorner \iota_1 \urcorner \boxplus \ulcorner \iota_2 \urcorner) \,;\, \ulcorner \nabla \urcorner \\
\sim & \quad \{\ \text{law (5.17)}\ \} \\
& \ulcorner \iota_1 + \iota_2 \urcorner \,;\, \ulcorner \nabla \urcorner \\
\sim & \quad \{\ \text{law (5.6)}\ \} \\
& \ulcorner (\iota_1 + \iota_2) \cdot \nabla \urcorner \\
\sim & \quad \{\ + \text{ absorption}\ \} \\
& \ulcorner [\iota_1, \iota_2] \urcorner \\
\sim & \quad \{\ + \text{ reflection}\ \} \\
& \ulcorner \mathsf{id}_{I+J} \urcorner \\
\sim & \quad \{\ \text{law (5.7)}\ \} \\
& \mathsf{copy}_{I+J}
\end{array}
$$

which establishes (5.24). For equation (5.25) consider,

$$
\begin{array}{ll}
& (p \boxplus q) \,;\, [p', q'] \\
\sim & \quad \{\ \text{definition of } \textit{either} \text{ in } \mathsf{Cp}\ \} \\
& (p \boxplus q) \,;\, ((p' \boxplus q') \,;\, \ulcorner \nabla \urcorner) \\
\sim & \quad \{\ ;\ \text{associative (5.5)}\ \} \\
& ((p \boxplus q) \,;\, (p' \boxplus q')) \,;\, \ulcorner \nabla \urcorner \\
\sim & \quad \{\ \boxplus \text{ functor (5.15)}\ \} \\
& ((p \,;\, p') \boxplus (q \,;\, q')) \,;\, \ulcorner \nabla \urcorner \\
\sim & \quad \{\ \text{definition of } \textit{either} \text{ in } \mathsf{Cp}\ \} \\
& [p \,;\, p', q \,;\, q']
\end{array}
$$

□

32. REMARK. As expected, the $\boxplus$ combinator can be written in terms of an *either* construction on components. In fact, for $p : I \longrightarrow O$ and $q : J \longrightarrow R$, we obtain

$$p \boxplus q \ \sim\ [p \,;\, \ulcorner \iota_1 \urcorner, p \,;\, \ulcorner \iota_2 \urcorner] \tag{5.26}$$

Proof.

$$\begin{array}{ll} & p \boxplus q \\ \sim & \quad \{ \text{ law (5.4) } \} \\ & (p \boxplus q) \,;\, \mathsf{copy}_{O+R} \\ \sim & \quad \{ \boxplus \text{ reflection (5.24) } \} \\ & (p \boxplus q) \,;\, [\ulcorner \iota_1 \urcorner, \ulcorner \iota_2 \urcorner] \\ \sim & \quad \{ \boxplus \text{ absorption (5.25) } \} \\ & [p \,;\, \ulcorner \iota_1 \urcorner, p \,;\, \ulcorner \iota_2 \urcorner] \end{array}$$

□

This leads to the basic observation that, once lifted to Cp, Set coproduct embeddings keep their naturality, *i.e.*,

$$\ulcorner \iota_1 \urcorner \,;\, (p \boxplus q) \;\sim\; p \,;\, \ulcorner \iota_1 \urcorner \tag{5.27}$$

$$\ulcorner \iota_2 \urcorner \,;\, (p \boxplus q) \;\sim\; q \,;\, \ulcorner \iota_2 \urcorner \tag{5.28}$$

Proof.

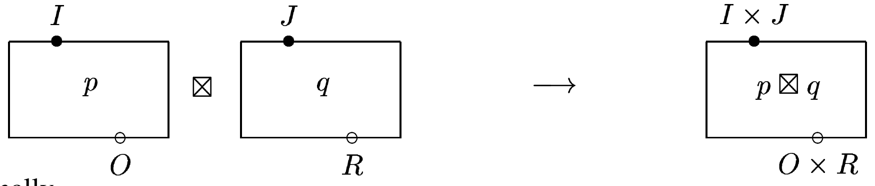
$$\begin{array}{ll} & \ulcorner \iota_1 \urcorner \,;\, (p \boxplus q) \\ \sim & \quad \{ \text{ law (5.26) } \} \\ & \ulcorner \iota_1 \urcorner \,;\, [p \,;\, \ulcorner \iota_1 \urcorner, p \,;\, \ulcorner \iota_2 \urcorner] \\ \sim & \quad \{ \boxplus \text{ cancellation (5.22) } \} \\ & p \,;\, \ulcorner \iota_1 \urcorner \end{array}$$

□

A direct corollary of this fact is the following ‘idempotency’ result:

$$p \,;\, \ulcorner \iota_1 \urcorner \;\sim\; \ulcorner \iota_1 \urcorner \,;\, (p \boxplus p) \tag{5.29}$$

33. PARALLEL. After the study of the *choice* combinator we proceed to that of the *parallel* combinator. This corresponds to a synchronous product: both components are executed simultaneously when triggered by a pair of input values. Note, however, that the behaviour effect, captured by monad B, propagates. For example, if B can express component failure and one of the arguments fails, the product will fail as well. Considering components $p : I \longrightarrow O$ and $p : J \longrightarrow R$, this is pictured as follows



Formally,

34. DEFINITION. The parallel combinator is defined as a lax functor $\boxtimes : \mathsf{Cp} \times \mathsf{Cp} \longrightarrow \mathsf{Cp}$, consisting of

- An action on objects given by

$$I \boxtimes J \;=\; I \times J$$

- A family of functors

$$\boxtimes_{IOJR} : \mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R) \longrightarrow \mathsf{Cp}(I \times J, O \times R)$$

yielding

$$p \boxtimes q \;=\; \langle \langle u_p, u_q \rangle \in U_p \times U_q, \overline{a}_{p \boxtimes q} \rangle$$

where

$$\begin{aligned} a_{p\boxtimes q} \;=\; \quad & U_p \times U_q \times (I \times J) \xrightarrow{\;\mathsf{m}\;} U_p \times I \times (U_q \times J) \\ & \xrightarrow{\;a_p \times a_q\;} \mathsf{B}\,(U_p \times O) \times \mathsf{B}\,(U_q \times R) \\ & \xrightarrow{\;\delta_l\;} \mathsf{B}\,(U_p \times O \times (U_q \times R)) \\ & \xrightarrow{\;\mathsf{B}\,\mathsf{m}\;} \mathsf{B}\,(U_p \times U_q \times (O \times R)) \end{aligned}$$

and maps pairs of arrows $\langle h_1, h_2 \rangle$ into $h_1 \times h_2$.

35. LEMMA. Combinator $\boxtimes$ is a lax functor in Cp_{B}, for B a commutative monad. In particular, for any components p, q, p' and q',

$$(p \boxtimes p')\,;(q \boxtimes q') \;\sim\; (p\,;q) \boxtimes (p'\,;q') \tag{5.30}$$

$$\mathsf{copy}_{K \boxtimes K'} \;\sim\; \mathsf{copy}_K \boxtimes \mathsf{copy}_{K'} \tag{5.31}$$

Proof. The argument structure is similar to the one in the proof of Lemma §24. We first check that, for any sets I, O, J and R, $\boxtimes_{I,O,J,R}$ is a functor from $\mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R)$ to $\mathsf{Cp}(I \times J, O \times R)$. The functoriality conditions come, again, directly from product functoriality. It remains to be proved that, given $h : p \longrightarrow p'$ and $k : q \longrightarrow q'$, $h \times k$ is a Cp-morphism from $p \boxtimes q$ to $p' \boxtimes q'$. The proof can be found in [Appendix D, page 367].

Next one has check the existence, for each tuple $\langle I, I', K, K', O, O' \rangle$ of objects, of a natural transformation

$$\mathsf{m}_{\boxtimes} : ;_{I \times I', K \times K', O \times O'} \circ (\boxtimes_{I,K,I',K'} \times \boxtimes_{K,O,K',O'}) \Longrightarrow \boxtimes_{I,O,I',O'} \circ (;_{I,K,O} \times ;_{I',K',O'})$$

whose components $\mathsf{m}_{p,q,p',q'}$, for each $p : I \longrightarrow K$, $q : K \longrightarrow O$, $p' : I' \longrightarrow K'$ and $q' : K' \longrightarrow O'$ are Cp 2-cells, *i.e.*, seed preserving comorphisms from $(p \boxtimes q) \mathbin{;} (p' \boxtimes q')$ to $(p \mathbin{;} p') \boxtimes (q \mathbin{;} q')$. Similarly, for each pair $\langle K, K' \rangle$ of objects, we look for a natural transformation

$$\mathsf{u}_{\boxtimes} : \mathsf{copy}_{K \boxtimes K'} \Longrightarrow \boxtimes_{K,K',K,K'} \circ (\mathsf{copy}_K \times \mathsf{copy}_{K'})$$

$\mathsf{u}_{\boxtimes}$ must be a seed preserving comorphism connecting the carriers $U_{\mathsf{copy}_{K \boxtimes K'}} = \mathbf{1}$ to $U_{\mathsf{copy}_K \boxtimes \mathsf{copy}_{K'}} = \mathbf{1} \times \mathbf{1}$.

As before, we take each $\mathsf{m}_{p,q,p',q'}$ as the component $\mathsf{m}_{U_p,U_q,U_{p'},U_{q'}} : (U_p \times U_q) \times (U_{p'} \times U_{q'}) \longrightarrow (U_p \times U_{p'}) \times (U_q \times U_{q'})$ of Set natural isomorphism m, and $\mathsf{u}_{\boxtimes}$ as $\mathsf{r}_{\mathbf{1}}^{\circ} : \mathbf{1} \longrightarrow \mathbf{1} \times \mathbf{1}$. Therefore, naturality, seed preservation and coherence in both cases are immediately established. In [Appendix D, page 367] we prove the comorphism conditions. Note that the commutativity of B is essential to establish $\mathsf{m}_{p,q,p',q'}$ as a comorphism.

□

36. LEMMA. Let f and g be functions. Then

$$\ulcorner f \urcorner \boxtimes \ulcorner g \urcorner \sim \ulcorner f \times g \urcorner \tag{5.32}$$

Proof. [Appendix D, page 370].

□

37. LEMMA. Up to isomorphic wiring and assuming the underlying monad B is commutative, $\boxtimes$ is a symmetric tensor product in each hom-category, with idle as unit and nil as a zero element. Thus,

$$(p \boxtimes q) \boxtimes r \sim (p \boxtimes (q \boxtimes r))[\mathsf{a}, \mathsf{a}^{\circ}] \tag{5.33}$$

$$p \boxtimes q \sim (q \boxtimes p)[\mathsf{s}, \mathsf{s}] \tag{5.34}$$

$$\mathsf{idle} \boxtimes p \sim p[\mathsf{r}, \mathsf{r}^{\circ}] \tag{5.35}$$

$$\mathsf{nil} \boxtimes p \sim \mathsf{nil}[\mathsf{zl}, \mathsf{zl}^{\circ}] \tag{5.36}$$

Proof. The laws above match with the observation that canonical Set isomorphisms a, r, l and zl, once lifted to Cp, keep their naturality up to bisimulation. Bisimulation is established by exhibiting a Cp 2-cell connecting both sides of each equation. This can be found in [Appendix D, page 371].

□

38. REMARK. In the lemma above, commutativity of the underlying monad B is only required to prove law (5.34). Assuming this law, the 'left' versions of equations (5.35) and (5.36), *i.e.*,

$$p \boxtimes \mathsf{idle} \sim p[\mathsf{l}, \mathsf{l}^\circ] \tag{5.37}$$

$$p \boxtimes \mathsf{nil} \sim \mathsf{nil}[\mathsf{zr}, \mathsf{zr}^\circ] \tag{5.38}$$

arise directly. For example,

$$\begin{array}{ll}
 & p \boxtimes \mathsf{idle} \\
\sim & \qquad \{ \text{ law (5.34) } \} \\
 & (\mathsf{idle} \boxtimes p)[\mathsf{s}, \mathsf{s}] \\
\sim & \qquad \{ \text{ law (5.35) } \} \\
 & p[\mathsf{r}, \mathsf{r}^\circ][\mathsf{s}, \mathsf{s}] \\
\sim & \qquad \{ \text{ law (5.9) } \} \\
 & p[\mathsf{r} \cdot \mathsf{s}, \mathsf{s} \cdot \mathsf{r}^\circ] \\
\sim & \qquad \{ \text{ routine: } \mathsf{l} = \mathsf{r} \cdot \mathsf{s} \text{ and } \mathsf{l}^\circ = \mathsf{s} \cdot \mathsf{r}^\circ \ \} \\
 & p[\mathsf{l}, \mathsf{l}^\circ]
\end{array}$$

Direct proofs of these equations can also be given, along arguments similar to the ones used in the proof of the corresponding 'right versions' in §37. Therefore, their validity does not depend on the underlying monad being commutative.

39. IS SPLIT DEFINABLE? A construction dual to the one discussed in §28 is certainly definable for $\boxtimes$. For this purpose, let $p : I \longrightarrow O$ and $q : I \longrightarrow R$ be two components sharing a common input type I and define

$$\langle p, q \rangle \ = \ \ulcorner \Delta \urcorner ; (p \boxtimes q)$$

This construction, however, fails to be a real *split* as, in the general case, the following diagram does not commute up to bisimulation,

$$\begin{tikzcd}
 & I \arrow[dl, "p"'] \arrow[d, "\langle p,q \rangle"] \arrow[dr, "q"] & \\
O & O \boxtimes R \arrow[l, "\ulcorner \pi_1 \urcorner"] \arrow[r, "\ulcorner \pi_2 \urcorner"'] & R
\end{tikzcd}$$

To see why, consider one of the equations in the diagram, for example,

$$\langle p, q\rangle \;;\ulcorner \pi_1 \urcorner \sim p \tag{5.39}$$

and try to establish bisimilarity by checking if $\pi_1 : U_p \times U_q \longrightarrow U_p$ is a comorphism. A tentative calculation goes as follows:

$$\begin{array}{cl}
& \mathsf{B}(\pi_1 \times \mathsf{id}) \cdot a_{\langle p,q\rangle;\ulcorner \pi_1 \urcorner} \\
= & \qquad \{ \text{ definitions } \} \\
& \mathsf{B}(\pi_1 \times \pi_1) \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \vartriangle) \\
= & \qquad \{ \text{ routine: } \pi_1 \times \pi_1 = \pi_1 \cdot \mathsf{m} \} \\
& \mathsf{B}\pi_1 \cdot \mathsf{Bm} \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \vartriangle) \\
= & \qquad \{ \mathsf{m}^\circ = \mathsf{m} \} \\
& \mathsf{B}\pi_1 \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \vartriangle) \\
= & \qquad \{ \star \} \\
& \pi_1 \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \vartriangle) \\
= & \qquad \{ \times \text{ cancellation } \} \\
& a_p \cdot \pi_1 \cdot \mathsf{m} \cdot (\mathsf{id} \times \vartriangle) \\
= & \qquad \{ \text{ routine: } \pi_1 \times \pi_1 = \pi_1 \cdot \mathsf{m} \} \\
& a_p \cdot (\pi_1 \times \pi_1) \cdot (\mathsf{id} \times \vartriangle) \\
= & \qquad \{ \text{ routine: } \pi_1 \cdot \vartriangle = \mathsf{id} \} \\
& a_p \cdot (\pi_1 \times \mathsf{id})
\end{array}$$

The problem arises in the step marked with a $\star$, because equality $\mathsf{B}\pi_1 \cdot \delta_l = \pi_1$ does not hold in general. It clearly holds for the identity and fails for B a non commutative monad. Taking B as the powerset monad, the equality holds unless the left argument is the empty set: for example, for non empty X, $(\mathsf{B}\pi_1 \cdot \delta_l)\langle X, \emptyset\rangle = \emptyset \neq X = \pi_1\langle X, \emptyset\rangle$. However, the diagram above commutes for the non empty powerset monad. This expresses a nondeterministic behaviour which excludes the possibility of *failure*.

This seems to be the general rule concerning the existence of *splits* for components based on commutative monads: the exclusion of failure. Intuitively, equation (5.39) fails because the eventual failure of q propagates, leading to the failure of $\langle p, q\rangle$.

It should be stressed that even in cases where cancellation fails (and consequently, construction $\langle p, q\rangle$ can hardly be called a *split*) the following *reflection* and *absorption* laws hold, the latter only in Cp_{B} for B a commutative monad:

$$\langle \ulcorner \pi_1 \urcorner, \ulcorner \pi_2 \urcorner \rangle \sim \mathsf{copy}_{O \times R} \tag{5.40}$$

$$\langle p, q \rangle \,;\, (p' \boxtimes q') \sim \langle p \,;\, p', q \,;\, q' \rangle \tag{5.41}$$

Proof. Let us check (5.40) in the first place:

$$\begin{array}{ll}
 & \langle \ulcorner \pi_1 \urcorner, \ulcorner \pi_2 \urcorner \rangle \\
\sim & \qquad \{ \text{ definition } \} \\
 & \ulcorner \triangle \urcorner \,;\, (\ulcorner \pi_1 \urcorner \boxtimes \ulcorner \pi_2 \urcorner) \\
\sim & \qquad \{ \text{ law (5.32) } \} \\
 & \ulcorner \triangle \urcorner \,;\, \ulcorner \pi_1 \times \pi_2 \urcorner \\
\sim & \qquad \{ \text{ law (5.6) } \} \\
 & \ulcorner (\pi_1 \times \pi_2) \cdot \triangle \urcorner \\
\sim & \qquad \{ \times \text{ absorption and identity in } \mathsf{Cp} \} \\
 & \mathsf{copy}_{O \times R}
\end{array}$$

Concerning (5.41):

$$\begin{array}{ll}
 & \langle p, q \rangle \,;\, (p' \boxtimes q') \\
\sim & \qquad \{ \text{ definition } \} \\
 & \ulcorner \triangle \urcorner \,;\, (p \boxtimes q) \,;\, (p' \boxtimes q') \\
\sim & \qquad \{ \text{ laws (5.5) and (5.30) } \} \\
 & \ulcorner \triangle \urcorner \,;\, ((p \,;\, p') \boxtimes (q \,;\, q')) \\
\sim & \qquad \{ \text{ definition } \} \\
 & \langle p \,;\, p', q \,;\, q' \rangle
\end{array}$$

Notice that law (5.30), used in the last proof, requires B to be commutative.

□

40. LIFTING $\triangle$. Recall from §29 that the *either* construction is definable in every bicategory of components, but, once lifted to the corresponding category of behaviours, fails the universal property. The reason is that the codiagonal arrow does not keep its naturality when lifted to Cp_{B}. Dually, let us look at the diagonal case. Should naturality be preserved the following equation would hold in Cp:

$$\ulcorner \triangle \urcorner \,;\, (p \boxtimes p) \sim p \,;\, \ulcorner \triangle \urcorner \tag{5.42}$$

The obvious candidate to establish bisimilarity is $\triangle: U_p \longrightarrow U_p \times U_p$, which is clearly seed preserving. The following calculation would prove the comorphism condition if it were not the case that, again, the step marked with a $\star$ is not valid in general.

$$\begin{array}{ll}
 & a_{(p \boxtimes p)} \cdot (\triangle \times \triangle) \\
= & \quad \{ \ \boxtimes \text{ definition} \} \\
 & \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_p) \cdot \mathsf{m} \cdot (\triangle \times \triangle) \\
= & \quad \{ \text{ routine: } \mathsf{m} \cdot (\triangle \times \triangle) = \triangle \} \\
 & \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_p) \cdot \triangle \\
= & \quad \{ \ \triangle \text{ natural} \} \\
 & \mathsf{Bm} \cdot \delta_l \cdot \triangle \cdot a_p \\
= & \quad \{ \ \star \ \} \\
 & \mathsf{Bm} \cdot \mathsf{B} \triangle \cdot a_p \\
= & \quad \{ \text{ routine: } \mathsf{m} \cdot \triangle = \triangle \times \triangle \} \\
 & \mathsf{B}(\triangle \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \triangle) \cdot a_p \\
= & \quad \{ \text{ definition} \} \\
 & \mathsf{B}(\triangle \times \mathsf{id}) \cdot a_{p;\ulcorner \triangle \urcorner}
\end{array}$$

In fact, $\delta_l \cdot \triangle = \mathsf{B} \triangle$ does not hold for any monad involving the notion of a collection. As a counter example compute $(\delta_l \cdot \triangle)\{5,2\} = \{\{5,5\},\{5,2\},\{2,5\},\{2,2\}\}$ and $(\mathcal{P}\triangle)\{5,2\} = \{\{5,5\},\{2,2\}\}$. On the other hand, it can be easily checked that (5.42) holds for the identity or the *maybe* monad, while the corresponding law for ∇ never holds. This means that the existence of a *split* construction satisfying cancellation depends crucially on B. The discussion above is summed up in the following results.

41. LEMMA. For B such that the lifting of $\triangle$ to $\mathsf{Cp_B}$ preserves naturality, the *split* construction defined in §39 obeys to the following *fusion* law:

$$r \,;\, \langle p, q \rangle \ \sim \ \langle r \,;\, p, r \,;\, q \rangle \tag{5.43}$$

Proof.

$$\begin{array}{ll}
 & r \,;\, \langle p, q \rangle \\
\sim & \quad \{ \text{ definition} \} \\
 & r \,;\, (\ulcorner \triangle \urcorner \,;\, (p \boxtimes q))
\end{array}$$

$$
\begin{array}{ll}
\sim & \quad \{ \text{ law (5.5) } \} \\
& (r \,;\, \ulcorner \triangle \urcorner) \,;\, (p \boxtimes q) \\
\sim & \quad \{ \text{ assumption (5.42) and law (5.5) again } \} \\
& \ulcorner \triangle \urcorner \,;\, ((r \boxtimes r) \,;\, (p \boxtimes q)) \\
\sim & \quad \{ \text{ law (5.30) } \} \\
& \ulcorner \triangle \urcorner \,;\, ((r \,;\, p) \boxtimes (r \,;\, q)) \\
\sim & \quad \{ \text{ definition } \} \\
& \langle r \,;\, p, r \,;\, q \rangle
\end{array}
$$

□

Note that this result holds even where $\boxtimes$ cancellation fails — which is typically the case of the bicategory of *partial* components.

42. LEMMA. If $\boxtimes$ cancellation holds in Cp_B and the lifting of $\triangle$ preserves naturality, then the $\boxtimes$ combinator lifts to a product in Bh_B. This is the case when B is the identity monad. Therefore, $\boxtimes$ lifts to a product in the category of behaviours of deterministic components.

Proof. Let us suppose that $\langle p, q \rangle$ is definable such that $\langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner \sim p$ and $\langle p, q \rangle \,;\, \ulcorner \pi_2 \urcorner \sim q$ and that there exists another component r satisfying the same equalities. Then, by $\sim$ transitivity, $r \,;\, \ulcorner \pi_1 \urcorner \sim \langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner$ and similarly for q and π_2. Thus

$$
\begin{array}{ll}
& \langle p, q \rangle \\
\sim & \quad \{ \boxtimes \text{ cancellation (5.39) } \} \\
& \langle \langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner, \langle p, q \rangle \,;\, \ulcorner \pi_2 \urcorner \rangle \\
\sim & \quad \{ \text{ assumption } \} \\
& \langle r \,;\, \ulcorner \pi_1 \urcorner, r \,;\, \ulcorner \pi_2 \urcorner \rangle \\
\sim & \quad \{ \boxtimes \text{ fusion (5.43) } \} \\
& r \,;\, \langle \ulcorner \pi_1 \urcorner, \ulcorner \pi_2 \urcorner \rangle \\
\sim & \quad \{ \boxtimes \text{ reflection (5.40) } \} \\
& r \,;\, \mathsf{copy}_{O \times R} \\
\sim & \quad \{ \text{ law (5.4) } \} \\
& r
\end{array}
$$

□

43. REMARK. Just as $\boxplus$ can be expressed in terms of the *either* construction (§32), the parallel combinator satisfies

$$p \boxtimes q \;\sim\; \langle \ulcorner \pi_1 \urcorner \,;\, p, \ulcorner \pi_2 \urcorner \,;\, q \rangle \tag{5.44}$$

provided B is commutative, even if $\boxtimes$ cancellation fails.

Proof.

$$\begin{array}{cl}
 & \langle p, q \rangle \\
\sim & \qquad \{ \text{ law (5.4) } \} \\
 & \mathsf{copy}_{I \times J} \,;\, \langle p, q \rangle \\
\sim & \qquad \{ \boxtimes \text{ reflection (5.40) } \} \\
 & \langle \ulcorner \pi_1 \urcorner, \ulcorner \pi_2 \urcorner \rangle \,;\, \langle p, q \rangle \\
\sim & \qquad \{ \boxtimes \text{ absorption (5.41) } \} \\
 & \langle \ulcorner \pi_1 \urcorner \,;\, p, \ulcorner \pi_2 \urcorner \,;\, q \rangle
\end{array}$$

□

Note, however, that Set product projections, once lifted to Cp, keep naturality only if $\boxtimes$ cancellation holds:

$$\begin{array}{cl}
 & \langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner \\
\sim & \qquad \{ \text{ law (5.44) } \} \\
 & \langle \ulcorner \pi_1 \urcorner \,;\, p, \ulcorner \pi_2 \urcorner \,;\, q \rangle \,;\, \ulcorner \pi_1 \urcorner \\
\sim & \qquad \{ \boxtimes \text{ cancellation (5.39) } \} \\
 & \ulcorner \pi_1 \urcorner \,;\, p
\end{array}$$

There is a simple, yet useful, special case: projection naturality is preserved wherever one of the $\boxtimes$ arguments is the lifting of a function, because $\ulcorner f \urcorner$ always exhibits a 'deterministic and total' behaviour embedded in the B structure, no matter how special such a structure is. Formally,

$$(\ulcorner f \urcorner \boxtimes q) \,;\, \ulcorner \pi_2 \urcorner \;\sim\; \ulcorner \pi_2 \urcorner \,;\, q \tag{5.45}$$

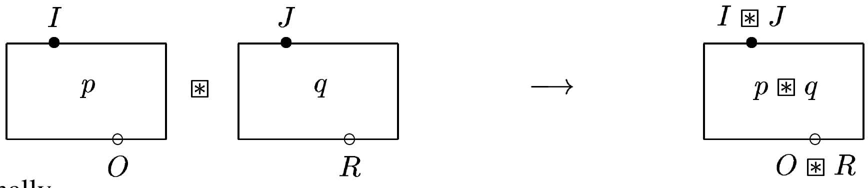
$$(p \boxtimes \ulcorner f \urcorner) \,;\, \ulcorner \pi_1 \urcorner \;\sim\; \ulcorner \pi_1 \urcorner \,;\, p \tag{5.46}$$

Proof. To verify (5.45), we just check that $\mathsf{r} : (\mathbf{1} \times U_q) \times \mathbf{1} \longrightarrow \mathbf{1} \times U_q$ is a comorphism. The proof of (5.46) is similar, but explicitly requires the commutativity of B.

$$
\begin{array}{ll}
 & \mathsf{B}(\mathsf{r} \times \pi_2) \cdot a_{\ulcorner f \urcorner \boxtimes q} \\
\sim & \qquad \{ \text{ definitions of } \ulcorner f \urcorner \text{ and } \boxtimes \} \\
 & \mathsf{B}(\mathsf{r} \times \pi_2) \cdot \mathsf{Bm} \cdot \delta_l \cdot ((\eta \cdot (f \times \mathsf{id})) \times a_q) \cdot \mathsf{m} \\
\sim & \qquad \{ \text{ routine: } (\mathsf{r} \times \pi_2) \cdot \mathsf{m} = \pi_2 \text{ and } \delta_l \text{ definition } \} \\
 & \mathsf{B}\pi_2 \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot ((\eta \cdot (f \times \mathsf{id})) \times a_q) \cdot \mathsf{m} \\
\sim & \qquad \{ \ \mu \text{ natural (C.16) } \} \\
 & \mu \cdot \mathsf{BB}\pi_2 \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot ((\eta \cdot (f \times \mathsf{id})) \times a_q) \cdot \mathsf{m} \\
\sim & \qquad \{ \text{ laws (C.18) and (C.13) } \} \\
 & \mu \cdot \mathsf{B}\pi_2 \cdot \eta \cdot ((f \times \mathsf{id}) \times a_q) \cdot \mathsf{m} \\
\sim & \qquad \{ \ \eta \text{ natural (C.17) } \} \\
 & \mu \cdot \eta \cdot \pi_2 \cdot ((f \times \mathsf{id}) \times a_q) \cdot \mathsf{m} \\
\sim & \qquad \{ \ \times \text{ cancellation } \} \\
 & \mu \cdot \eta \cdot a_q \cdot \pi_2 \cdot \mathsf{m} \\
\sim & \qquad \{ \text{ monad unit (C.14) } \} \\
 & a_q \cdot \pi_2 \cdot \mathsf{m} \\
\sim & \qquad \{ \text{ routine: } (\mathsf{r} \times \pi_2) = \pi_2 \cdot \mathsf{m} \} \\
 & a_q \cdot (\mathsf{r} \times \pi_2)
\end{array}
$$

□

44. Concurrent Components. The last combinator is *concurrent* composition, denoted by $\boxast$. It combines choice and parallel, in the sense that p and q can be executed independently or jointly, depending on the input supplied. The corresponding diagram is



Formally,

45. Definition. Concurrent composition is defined as a lax functor $\boxast : \mathsf{Cp} \times \mathsf{Cp} \longrightarrow \mathsf{Cp}$, consisting of

- An action on objects given by

$$I \boxast J \;=\; I + J + I \times J$$

- A family of functors

$$\boxast_{IOJR} : \mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R) \longrightarrow \mathsf{Cp}(I + J + I \times J, O + R + O \times R)$$

yielding

$$p \boxast q \;=\; \langle \langle u_0, v_0 \rangle \in U_p \times U_q, \overline{a}_{p \boxast q} \rangle$$

where $a_{p \boxast q}$ is defined as:

$$\begin{aligned} U_p \times U_q \times (I \boxast J) &\xrightarrow{\mathsf{dr}} U_p \times U_q \times (I + J) + U_p \times U_q \times (I \times J) \\ &\xrightarrow{a_{p \boxplus q} + a_{p \boxtimes q}} \mathsf{B}\,(U_p \times U_q \times (O + R)) + \mathsf{B}\,(U_p \times U_q \times (O \times R)) \\ &\xrightarrow{[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]} \mathsf{B}\,(U_p \times U_q \times (O \boxast R)) \end{aligned}$$

and mapping pairs of arrows $\langle h_1, h_2 \rangle$ into $h_1 \times h_2$.

46. Lemma. $\boxast$ is a lax functor in Cp_{B}, for B a commutative monad. In particular, for any components p, q, p' and q',

$$(p \boxast p') \,;\, (q \boxast q') \;\sim\; (p \,;\, q) \boxast (p' \,;\, q') \tag{5.47}$$

$$\mathsf{copy}_{K \boxast K'} \;\sim\; \mathsf{copy}_K \boxast \mathsf{copy}_{K'} \tag{5.48}$$

Proof. The proof follows the argument used when proving corresponding results for $\boxplus$ and $\boxtimes$ (§§24 and 35). In particular, the verification that $\boxast$ is well defined on arrows (entailing, for any sets I, O, J and R, the definition of a functor $\boxast_{I,O,J,R}$ from $\mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R)$ to $\mathsf{Cp}(I \boxast J, O \boxast R)$) is straightforward once assumed identical results for $\boxplus$ and $\boxtimes$. Therefore, let us focus on the proof of equations (5.47) and (5.48). A proof 'from first principles' would exhibit the family of 'associative' and 'unit' natural transformations, as done for the other two tensors discussed above. Here again Set natural isomorphisms m and r° are the obvious choices. Thus, for example, the unit law (5.48) is established by the following calculation:

$$
\begin{aligned}
& a_{\mathsf{copy}_K \boxast \mathsf{copy}_{K'}} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= \quad & \{ \ \boxast \text{ definition } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{\mathsf{copy}_K \boxplus \mathsf{copy}_{K'}} + a_{\mathsf{copy}_K \boxtimes \mathsf{copy}_{K'}}) \cdot \mathsf{dr} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= \quad & \{ \ \text{laws (5.16) for } \boxplus \text{ and (5.31) for } \boxtimes \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{\mathsf{copy}_{K \boxplus K'}} + a_{\mathsf{copy}_{K \boxtimes K'}}) \cdot \mathsf{dr} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= \quad & \{ \ \text{identity definition in } \mathsf{Cp} \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\eta + \eta) \cdot \mathsf{dr} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= \quad & \{ \ \text{law (C.64)} \ \} \\
& \eta \cdot \mathsf{dr}^\circ \cdot \mathsf{dr} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= \quad & \{ \ \mathsf{dr} \text{ isomorphism } \} \\
& \eta \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= \quad & \{ \ \text{identity definition in } \mathsf{Cp} \ \} \\
& \mathsf{B}(\mathsf{r}^\circ \times \mathsf{id}) \cdot a_{\mathsf{copy}_{K \boxast K'}}
\end{aligned}
$$

Next we give a more direct proof of law (5.47), by explicitly re-using the proofs of the similar laws for ⊞ and ⊠. We start by unfolding the definitions:

$$
\begin{aligned}
& a_{(p \boxast p');(q \boxast q')} \\
= \quad & \{ \ ; \text{ definition } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q \boxast q'}) \cdot \mathsf{Ba} \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_{p \boxast p'} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \boxast \text{ definition } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{q \boxplus q'} + a_{q \boxtimes q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \\
& \cdot \mathsf{Ba} \cdot \mathsf{Bxr} \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot ((a_{p \boxplus p'} + a_{p \boxtimes p'}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
& \vdots \\
= \quad & \{ \ \text{details in [Appendix D, page 373]} \} \\
& \vdots \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu + \mu) \cdot (\mathsf{B}\tau_l + \mathsf{B}\tau_l) \cdot (\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'})) \\
& \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'}) \cdot (\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= \quad & \{ \ ; \text{ definition } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{(p \boxplus p');(q \boxplus q')} + a_{(p \boxtimes p');(q \boxtimes q')}) \cdot \mathsf{dr}
\end{aligned}
$$

Note that $a_{(p\boxplus p');(q\boxplus q')}$ and $a_{(p\boxtimes p');(q\boxtimes q')}$ can be replaced, respectively, by $a_{(p;q)\boxplus(p';q')}$ and $a_{(p;q)\boxtimes(p';q')}$, according to laws (5.15) in §5.24 and (5.30) in §5.35. The substitution is, of course, up to bisimulation. And, in particular, recall that law (5.30) in §5.35 assumes the behaviour monad B to be commutative. In any case, however, we may conclude $a_{(p;q)\boxast(p';q')}$, by the definition of $\boxast$ and up to bisimulation. The proof details are supplied in [Appendix D, page 373].

□

47. LEMMA. Up to isomorphic wiring and assuming the underlying monad B is commutative, $\boxast$ is a symmetric tensor product in each hom-category with nil as unit. Thus,

$$(p \boxast q) \boxast r \sim (p \boxast (q \boxast r))[\mathsf{a}_*, {\mathsf{a}_*}^\circ] \tag{5.49}$$

$$p \boxast q \sim (q \boxast p)[\mathsf{s}_*, \mathsf{s}_*] \tag{5.50}$$

$$\mathsf{nil} \boxast p \sim p[\mathsf{r}_*, {\mathsf{r}_*}^\circ] \tag{5.51}$$

$$p \boxast \mathsf{nil} \sim p[\mathsf{l}_*, {\mathsf{l}_*}^\circ] \tag{5.52}$$

where wiring resorts to the following Set natural isomorphisms

$$\begin{aligned}
\mathsf{a}_{*I,J,H} &: (I \boxast J) \boxast H \longrightarrow I \boxast (J \boxast H)\\
\mathsf{s}_{*I,J} &: (I \boxast J) \longrightarrow (J \boxast I)\\
\mathsf{l}_{*I} &: (I \boxast \emptyset) \longrightarrow I\\
\mathsf{r}_{*I} &: (\emptyset \boxast I) \longrightarrow I
\end{aligned}$$

Proof. As discussed in the proof of lemma §46, we are interested in re-using the proofs of corresponding results concerning $\boxplus$ and $\boxtimes$, which can be done in two possible ways. The simplest one proceeds by unfolding one of the sides of the equation until previous laws can be applied. Substitutions made at this point are, of course, up to a bisimulation. Then we proceed by equational reasoning until the expression on the other side of the equation is reached. In doing this we prove that both sides, as Cp arrows, are, not equal, but bisimilar. Nothing is said, however, about the particular comorphism which witnesses such bisimilarity.

Alternatively, such a comorphism can be identified —as a function connecting the corresponding state spaces —and proved to be seed preserving and actually a comorphism. This is the procedure we have taken in most proofs earlier on, when no previous component's laws were being re-used. The trouble in adopting it when re-use is desirable is that it requires not only to re-use the law but also its proof. More exactly, whenever re-using a bisimulation equation, the comorphism which witnesses it has to become explicit.

In [Appendix D, page 376] we illustrate both procedures in the detailed proof of equation (5.52). The other proofs are also sketched, although they offer no particular difficulty as similar results have already been established for both $\boxplus$ and $\boxtimes$. As expected, only equation (5.50) requires the commutativity of the underlying behaviour monad.

□

48. LEMMA. The intuition about the relationship between $\boxast$ and both $\boxplus$ and $\boxtimes$ is formally expressed by the following laws, for any components p and q:

$$\ulcorner\iota_1\urcorner \,;\, (p \boxast q) \;\sim\; (p \boxplus q) \,;\, \ulcorner\iota_1\urcorner \tag{5.53}$$

$$\ulcorner\iota_2\urcorner \,;\, (p \boxast q) \;\sim\; (p \boxtimes q) \,;\, \ulcorner\iota_2\urcorner \tag{5.54}$$

Proof. We prove both equalities on the equivalent formulation in terms of *wrapping*. Under such a formulation they appear as strict Cp arrow's equalities. Thus,

$$\begin{aligned}
& a_{(p \boxast q)[\iota_1,\mathsf{id}]} \\
= & \quad \{ \; \boxast \text{ and } \textit{wrapping} \text{ definitions} \; \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{(p \boxplus q)} + a_{(p \boxtimes q)}) \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_1) \\
= & \quad \{ \; \text{law (C.55)} \; \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{(p \boxplus q)} + a_{(p \boxtimes q)}) \cdot \iota_1 \\
= & \quad \{ \; + \text{ absorption} \; \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{(p \boxplus q)}, \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{(p \boxtimes q)}] \cdot \iota_1 \\
= & \quad \{ \; + \text{ cancellation} \; \} \\
& \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{(p \boxplus q)} \\
= & \quad \{ \; \boxast \text{ and } \textit{wrapping} \text{ definitions} \; \} \\
& a_{(p \boxplus q)}[\mathsf{id}, \iota_1]
\end{aligned}$$

and, similarly, for (5.54).

□

3. Interaction

49. INTERACTION. So far component interaction has been centred upon sequential composition (§2), which is the Cp counterpart to functional composition in Set. The basic interaction model is input to output connection. This section introduces two combinators which push forward such model. The first one, in §50, is *feedback*: the output of a component p is directly linked to its input thus forcing the feed back of the information produced. The second one (§51) is a *partial feedback* combinator which generalises sequential composition by connecting some input to some output wires and, consequently, forcing *part* of the output of a component to be fed back as input. Notice that both combinators are 'partial' in the sense that their definition is made in terms of functors among some families of Cp hom-categories.

50. FEEDBACK. The *feedback* combinator is defined, for each object Z, as a family of functors $\Lsh$: $\mathsf{Cp}(Z, Z) \longrightarrow \mathsf{Cp}(Z, Z)$ which is the identity on arrows and maps each component $p : Z \longrightarrow Z$ to

$$p \Lsh : Z \longrightarrow Z \;=\; \langle u_p \in U_p, \overline{a}_{p\Lsh} \rangle$$

where

$$a_{p\Lsh} \;=\; \qquad U_p \times Z \xrightarrow{a_p} \mathsf{B}(U_p \times Z) \xrightarrow{\mathsf{B}a_p} \mathsf{BB}(U_p \times Z) \xrightarrow{\mu} \mathsf{B}(U_p \times Z)$$

That is to say: $a_{p\Lsh} = a_p \bullet a_p$, recalling that $\bullet$ is the Kleisli composition (§A.7) for B. Note that, in general, $p \Lsh$ will not be bisimilar to $p \,;\, p$ (in the latter case each copy of p will maintain its own private state space).

51. PARTIAL FEEDBACK. The *partial feedback* combinator applies to components of type $I + Z \longrightarrow O + Z$, for I, O and Z arbitrary. The feed back component is executed for any input. Execution terminates if the result is of type O. Should it be of type Z, it is fed back and p runs again for that value. Formally, it is defined, for each tuple of objects I, O and Z, as a family of functors $- \Lsh_Z$: $\mathsf{Cp}(I + Z, O + Z) \longrightarrow \mathsf{Cp}(I + Z, O + Z)$ which are the identity on arrows and map each component $p : I + Z \longrightarrow O + Z$ to $p \Lsh_Z : I + Z \longrightarrow O + Z$ given by

$$p \Lsh_Z \;=\; \langle u_p \in U_p, \overline{a}_{p\Lsh_Z} \rangle$$

where

$$\begin{aligned} a_{p\Lsh_Z} \;=\; \qquad & U_p \times (I + Z) \xrightarrow{\qquad a_p \qquad} \mathsf{B}(U_p \times (O + Z)) \\ & \xrightarrow{\qquad \mathsf{B}\mathsf{dr} \qquad} \mathsf{B}(U_p \times O + U_p \times Z) \\ & \xrightarrow{\mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2)} \mathsf{B}(U_p \times (O + Z) + U_p \times (I + Z)) \\ & \xrightarrow{\mathsf{B}(\eta + a_p)} \mathsf{B}(\mathsf{B}(U_p \times (O + Z)) + \mathsf{B}(U_p \times (O + Z))) \\ & \xrightarrow{\qquad \mathsf{B}\nabla \qquad} \mathsf{BB}(U_p \times (O + Z)) \\ & \xrightarrow{\qquad \mu \qquad} \mathsf{B}(U_p \times (O + Z)) \end{aligned}$$

52. LEMMA. Both feedback combinators introduced in §§50, 51 are endofunctors in the suitable hom-categories.

Proof. As the action of both functors is the identity on arrows, the functoriality conditions hold trivially once proved that any arrow $h : p \longrightarrow q$, for p and q suitably typed, is still an arrow from $p \Lsh$ (respectively, $p \Lsh_Z$) to $q \Lsh$ (respectively, $q \Lsh_Z$). As seeds are strictly preserved in both cases, what remains to be shown is the comorphism condition for both combinators. This can be found in [Appendix D, page 380].

□

53. LEMMA. Both combinators specialise to components representing functions according to the following laws: let $f : Z \longrightarrow Z$ and $g : I + Z \longrightarrow O + Z$, then,

$$\ulcorner f \urcorner \Lsh \ \sim \ulcorner f \cdot f \urcorner \tag{5.55}$$

$$\ulcorner g \urcorner \Lsh_Z \ \sim \ulcorner [\iota_1, g \cdot \iota_2] \cdot g \urcorner \tag{5.56}$$

Proof. The proof can be found in [Appendix D, page 382]. Notice the intuition about equation (5.56) is that applying partial feedback to a function g lifted to Cp amounts to (the lifting of) a conditional composition of g with itself, depending on input.

□

54. LEMMA. Let $p : Z \longrightarrow Z$ and $q : I \longrightarrow O$ be components. Then

$$p \Lsh \ \sim p[\mathsf{r}_+, \mathsf{r}_+{}^\circ] \Lsh_Z \ [\mathsf{r}_+{}^\circ, \mathsf{r}_+] \tag{5.57}$$

$$q \boxplus p \Lsh \ \sim (q \boxplus p) \Lsh_Z \tag{5.58}$$

$$p \Lsh \boxplus q \sim (q \boxplus p) \Lsh_Z \ [\mathsf{s}_+, \mathsf{s}_+] \tag{5.59}$$

Proof. The proofs of both (5.57) and (5.58) are given in [Appendix D, page 383]. Notice that (5.59) is a direct consequence of (5.58) and $\boxplus$ commutativity (5.21).

□

55. REMARK. Note that all laws in §53 and §54, with the exception of (5.59) which uses a $\sim$ equation, are actually strict Cp arrow equalities, and not just bisimulations. Also notice that equation (5.58) generalises to

$$(q \boxplus p)[\mathsf{a}_+, \mathsf{a}_+{}^\circ] \Lsh_Z \ \sim (q \boxplus p \Lsh_Z)[\mathsf{a}_+, \mathsf{a}_+{}^\circ] \tag{5.60}$$

for $p : J + Z \longrightarrow R + Z$.

56. The following two lemmas relate feedback combinators with the other operators in the algebra. Note that all equations are strict Cp arrow equalities. However, validity of (5.63) and (5.64) will depend on the commutativity of the behaviour monad B. Also note that there are no laws relating partial feedback to either $\boxtimes$ or $\boxast$, as shown by an inspection of the relevant interface structures.

57. Lemma. Let $p : Z \longrightarrow Z$ and $q : R \longrightarrow R$ be components and $i : W \longrightarrow Z$ be a Set isomorphism. Then

$$p \Lsh [i, i^\circ] \sim p[i, i^\circ] \Lsh \tag{5.61}$$

$$(p \boxplus q) \Lsh \ \sim p \Lsh \boxplus q \Lsh \tag{5.62}$$

$$(p \boxtimes q) \Lsh \ \sim p \Lsh \boxtimes q \Lsh \tag{5.63}$$

$$(p \boxast q) \Lsh \ \sim p \Lsh \boxast q \Lsh \tag{5.64}$$

Proof. [Appendix D, page 387].

□

58. Lemma. Let $p : I{+}K \longrightarrow O{+}K$ and $q : J \longrightarrow R$ be components, $f : I' \longrightarrow I$, $g : O \longrightarrow O'$ functions and $i : W \longrightarrow K$ a Set isomorphism. Then

$$p \Lsh_K [f + i, g + i^\circ] \sim p[f + i, g + i^\circ] \Lsh_W \tag{5.65}$$

$$(p \boxplus q)[\mathsf{xr}_+, \mathsf{xr}_+] \Lsh_K \ \sim (p \Lsh_K \boxplus q)[\mathsf{xr}_+, \mathsf{xr}_+] \tag{5.66}$$

Proof. [Appendix D, page 389].

□

4. Some Examples

59. The Game of Life. This section provides a few examples which illustrate the component model developed in this chapter. The first one is the *game of life*, a simple model of cellular behaviour which has been popularised as a common screen locker for computers. The game is based on a grid of cells each of which sends and receives elementary stimulus to and from its four adjacent neighbours. A stimulus is a Boolean value indicating whether the cell is either ‘alive’ or ‘dead’. The following few rules govern the survival, death and birth of cell generations:

- Each living cell with less than two or more than three living neighbours dies in the next generation.
- Each dead cell with exactly three living neighbours becomes alive.
- Each living cell with less than two or three living neighbours survives until the next generation.

Each cell will be specified as a component Cell whose input is a tuple of four Boolean values, each one to be supplied by one of the four adjacent cells. The cell reacts to such a stimulus by computing its new state — 'dead' or 'alive' — and making it available as an output to its neighbours, used to compute the next cell generation. Formally, we define

$$\mathsf{Cell} : \mathbf{2} \times \mathbf{2} \times \mathbf{2} \times \mathbf{2} \longrightarrow \mathbf{2} \quad = \quad \langle \mathsf{true} \in \mathbf{2}, \overline{a}_{\mathsf{Cell}} \rangle$$

where

$$a_{\mathsf{Cell}}\ \langle u, t\rangle \ = \ \text{let } n = \mathsf{living}\, t \\ \text{in } \begin{cases} \langle \mathsf{false}, \mathsf{false} \rangle & \text{if } u = \mathsf{true} \wedge (n < 1 \vee n > 3) \\ \langle \mathsf{true}, \mathsf{true} \rangle & \text{if } u = \mathsf{false} \wedge n = 3) \\ \langle u, u \rangle & \text{otherwise} \end{cases}$$

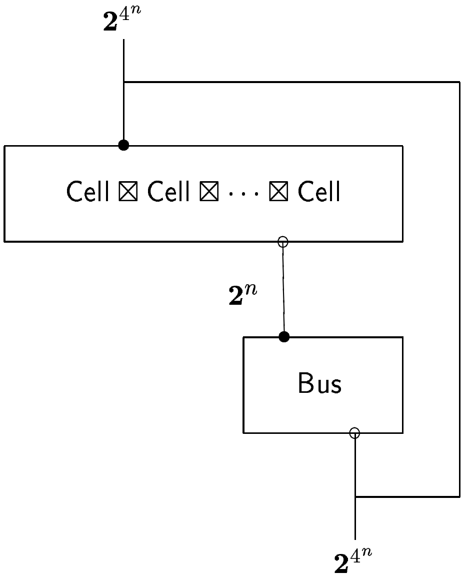
Function living above, counts the number of living stimuli (*i.e.*, the number of true values) in a four Boolean tuple. So, $U_{\mathsf{Cell}} = \mathbf{2}$ and $\mathsf{B} = \mathsf{Id}$.

The game's behaviour is, of course, deterministic and all cells in the grid react simultaneously to produce the new generation. To form a grid of n cells we simply connect them using the *parallel* combinator ⊠. The crucial point is to devise a wiring scheme to guarantee that the joint output of the n connected cells is appropriately fed back. The composed system is pictured below, where component

$$\mathsf{Bus} : \mathbf{2}^n \longrightarrow \mathbf{2}^{4^n}$$

concentrates and correctly distributes the output.

The n cells are organised as a fully connected matrix of k rows and l columns ($n = k \times l$), so that the neighbours of cell $\langle i, j\rangle$ are $\langle i-1, j\rangle$, $\langle i+1, j\rangle$, $\langle i, j-1\rangle$ and $\langle i, j+1\rangle$ (in the 'west', 'east', 'north' and 'south' directions, respectively) computed in the k and l rings (*i.e.*, $1-1 = k$, $k+1 = 1$ and $1-1 = l$, $l+1 = 1$).



To specify Bus we adopt the following convention: the first cell in the ⊠-expression has coordinates $\langle 1, 1\rangle$, second is $\langle 1, 2\rangle$ and so on until column n is reached; the next cell is then $\langle 2, 1\rangle$. Under this convention the output produced by cell $\langle i, j\rangle$ is selected from the global output tuple as the $j + (n \times (i - 1))$-projection, *i.e.*

$$\begin{aligned} \mathsf{out}_{\langle i,j\rangle} &: \mathbf{2}^n \longrightarrow \mathbf{2} \\ \mathsf{out}_{\langle i,j\rangle} &= \pi_{j+(n\times(i-1))} \end{aligned}$$

Now, the input to cell $\langle i, j\rangle$ is simply the *split* of the outputs of its neighbours, *i.e.*,

$$\begin{aligned} \mathsf{in}_{\langle i,j\rangle} &: \mathbf{2}^n \longrightarrow \mathbf{2}^4 \\ \mathsf{in}_{\langle i,j\rangle} &= \langle \mathsf{out}_{\langle i,\mathsf{dec}_n j\rangle}, \mathsf{out}_{\langle \mathsf{dec}_n i,j\rangle}, \mathsf{out}_{\langle i,\mathsf{inc}_n j\rangle}, \mathsf{out}_{\langle \mathsf{inc}_n i,j\rangle} \rangle \end{aligned}$$

where $\mathsf{dec}_n x = (x = 1 \to n, x - 1)$ and $\mathsf{inc}_n x = (x = n \to 1, x + 1)$. Finally, Bus is defined as the lifting of *split*

$$\mathsf{w} = \langle \mathsf{in}_{\langle i,j\rangle} \mid i, j \in 1..n \rangle$$

Finally,

$$\mathsf{GameLife} = ((\mathsf{Cell} \boxtimes \mathsf{Cell} \boxtimes \cdots \boxtimes \mathsf{Cell}) \,;\, \mathsf{Bus}) \Lsh$$

where

$$\mathsf{Bus} = \ulcorner \mathsf{w} \urcorner$$

Clearly, by law (5.10), $\mathsf{GameLife} \sim ((\mathsf{Cell} \boxtimes \mathsf{Cell} \boxtimes \cdots \boxtimes \mathsf{Cell})[\mathsf{id}, \mathsf{w}])\ {}^{\Lsh}$. In any case, the *feedback* combinator is responsible for extending the game's behaviour to the infinite, once the component has been stimulated with an initial input.

60. THE WIRING PROBLEM. Consider a component $p : I + J \longrightarrow O \times R$. Liftings of canonical Set injections and projections provide a simple way to restrict access to p. For example

$$\ulcorner \iota_1 \urcorner \,;\, p \,;\, \ulcorner \pi_2 \urcorner$$

restricts the use of p to the first input, while selecting only the second factor of its output. Notice, however, that the first factor of the output is lost.

On the other hand, in an additive output situation unused parcels cannot be hidden but simply collapsed into, say, the trivial type $\mathbf{1}$. For example, if p's output is $O + R$, the 'best' we can get is $O + \mathbf{1}$ by wrapping p as $p[\mathsf{id}, \mathsf{id}+!]$. Output of type $\mathbf{1}$ may be interpreted as a signal of some activity going on (*e.g.*, the turning of a led), but, in general, p cannot be forced to suppress it. In the next section we shall discuss a particular class of components in which a proper restriction can be enforced. For the moment, however, the next two paragraphs discuss two common wiring situations which require the introduction of auxiliary components.

61. MERGING. Suppose a component $p : I \times J \longrightarrow O$, which requires a pair of input stimuli, is to be used in a situation where I and J values are produced by two different sources. The idea is to guarantee that alternative inputs get packed and appropriately supplied to p. Packing, however, introduces a time dimension as typically both inputs will not be simultaneously available. Therefore we specify a merging component whose state space is $C_{I,J} = I \times J + I + J + \mathbf{1}$, depending on both inputs being already known, and thus ready to be forward to p, or only one of them, or even none, being so. In the latter cases the output of Merge would simply be $\mathbf{1}$. The fact that $C_{I,J}$ is isomorphic to $(I + \mathbf{1}) \times (J + \mathbf{1})$ suggests the following specification:

$$\mathsf{Merge} : I + J \longrightarrow \mathbf{1} + (I \times J) \quad = \quad \langle \langle \iota_2 *, \iota_2 * \rangle \in (I + \mathbf{1}) \times (J + \mathbf{1}), \overline{a}_{\mathsf{Merge}} \rangle$$

where

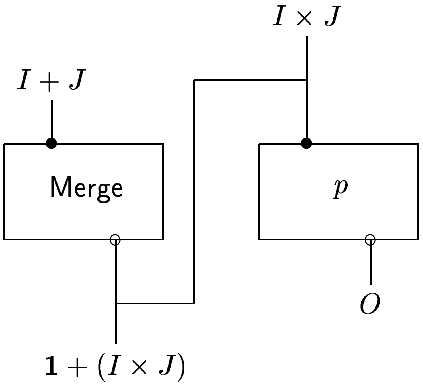
$$a_{\mathsf{Merge}}\ \langle u, x \rangle \ = \eta_{\mathsf{B}} \begin{cases} \langle \langle x, (\pi_2\ u) \rangle, ((\text{is--}J\ \pi_2 u'\ ) \ \rightarrow\ (\iota_2\ u'),\ (\iota_1\ *)) \rangle & \text{if } (\text{is--}I\ x) \\ \langle \langle (\pi_1\ u), x \rangle, ((\text{is--}I\ \pi_2 u'\ ) \ \rightarrow\ (\iota_2\ u'),\ (\iota_1\ *)) \rangle & \text{if } (\text{is--}J\ x) \end{cases}$$

where, following a well established convention, we denote the updated value of the state variable u by u' and make use of injection predicates — $(\text{is--}T\ \ x)$ — popularised in the VDM meta-language [Jon83]. The inclusion of the monad unit η makes the definition generic for the class of components we have been dealing within this chapter.

By aggregating Merge and p, the relevant stimuli will be passed to p whenever possible. This is achieved by *partial feedback* which, on its turn, requires additional wiring to be applied. The result of such a composition is

$$((\mathsf{Merge} \boxplus p)\,[\mathsf{id}, \mathsf{xr}_+])\, \Lsh_{I \times J}$$

cf.,

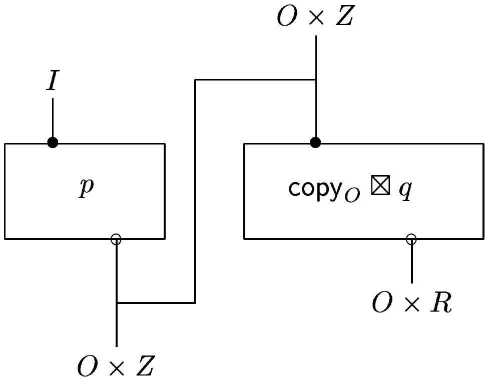


The type of $\mathsf{Merge} \boxplus p$ is $(I + J) + (I \times J) \longrightarrow (\mathbf{1} + (I \times J)) + O$. Thus the need for the xr_+ output wrapping to fit the typing of the *partial feedback* combinator.

62. DELAYED PROJECTIONS. The other common situation mentioned in §60 concerns the lost of part of a multiplicative output. Given $p : I \longrightarrow O \times Z$ and $q : Z \longrightarrow R$, should p and q be linked as in

$$p \,;\, \ulcorner \pi_2 \urcorner \,;\, q : I \longrightarrow R$$

the O typed p output will not be re-usable any more. A simple wiring scheme, referred to as a *delayed projection*, will prevent this situation by propagating the unused output along the composition. This is achieved by packing q with an identity component for O, *i.e.*,



Formally,

$$p \,;\, (\mathsf{copy}_O \boxtimes q)$$

The wiring correctness, *i.e.*, the fact that both components exhibit the same behaviour if the O output is to be discarded, is easily established:

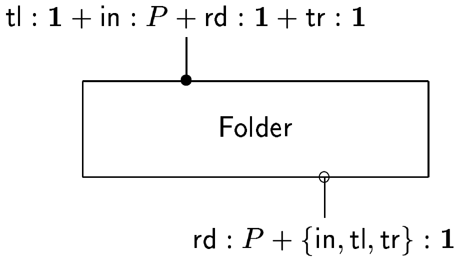
$$\begin{array}{ll} & (p \,;\, (\mathsf{copy}_O \boxtimes q)) \,;\, \ulcorner \pi_2 \urcorner \\ \sim & \quad \{ \text{ law (5.5) } \} \\ & p \,;\, ((\mathsf{copy}_O \boxtimes q) \,;\, \ulcorner \pi_2 \urcorner) \\ \sim & \quad \{ \text{ law (5.45) } \} \\ & p \,;\, (\ulcorner \pi_2 \urcorner \,;\, q) \\ \sim & \quad \{ \text{ law (5.5) } \} \\ & (p \,;\, \ulcorner \pi_2 \urcorner) \,;\, q \end{array}$$

63. A Folder From Two Stacks. The purpose of the next example is to illustrate how new components can be built from old ones, relying solely on the old functionality available. The example is the construction of a *folder* from two *stack* components. Although these components are parametric on the type of stacked objects, we will refer to these as ‘pages’, by analogy with a folder in which new ‘pages’ are inserted on the righthandside pile and retrieved (‘read’) from the lefthandside one.

A static, VDM-like specification of the component we have in mind can be found in [Oli92a]. According to this specification, the Folder component should provide operations to *read*, *insert* a new page, *turn a page right* and *turn a page left*. Reading returns the page which one can read once the folder is open at some position. Insertion takes as argument the page to be inserted. The other two operations are simply state updates. Let P be the type of a *page*. The Folder buttons, or actions, have the following signatures

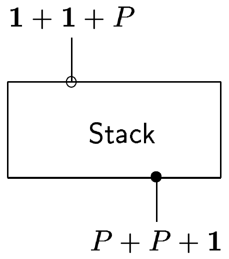
$$\begin{cases} \mathsf{in}: & P \longrightarrow \mathbf{1} \\ \mathsf{rd}: & \mathbf{1} \longrightarrow P \\ \mathsf{tl}, \mathsf{tr}: & \mathbf{1} \longrightarrow \mathbf{1} \end{cases}$$

which may be represented in the following picture, where input and output types are decorated with the corresponding action names:


$\mathsf{tl} : \mathbf{1} + \mathsf{in} : P + \mathsf{rd} : \mathbf{1} + \mathsf{tr} : \mathbf{1}$
Folder
$\mathsf{rd} : P + \{\mathsf{in}, \mathsf{tl}, \mathsf{tr}\} : \mathbf{1}$

Now, think of a pre-existing component Stack, modelling the *stack of P* data type and offering the usual operations:

$$\begin{cases} \mathsf{top}: & \mathbf{1} \longrightarrow P \\ \mathsf{pop}: & \mathbf{1} \longrightarrow P \\ \mathsf{push}: & P \longrightarrow \mathbf{1} \end{cases}$$



The picture on the right avoids decorations by ‘adding’ input and output types from left to right in the same order as they appear in the signature on the left, starting from top. A stack can be modelled as a *partial* component, letting the *maybe* monad capture exceptions, but we shall not be concerned with its ‘internals’ here — just assume it behaves as it is expected to.

Our exercise is to build Folder using two stacks to model of the *left* and *right* piles of pages, respectively. The intuition is that the push action of the right stack will be connected to the pop of the left one to model tr, the ‘turn page right’ action, and that a symmetric connection be used to model tl. The rd operation consumes the ‘front’ page — the one which can be accessed by top on the left stack. Clearly, action top on the right stack will not be used. On the other hand, two roles are assigned to its push

action: to be part of the tr realisation, as mentioned above, and to directly model page insertion into the folder, *i.e.*, action in.

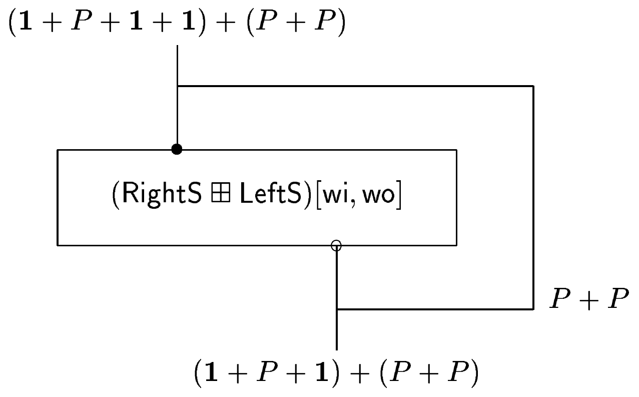
According to this plan, the assembly of the Folder starts by defining RightS as a Stack component suitably wrapped to meet the above mentioned constraints. At the input level we need to hide access to the top button, resorting to ι_2, and replicate the input to push by wrapping p with the codiagonal ∇_P. At the output level, because of its additive structure, we cannot get rid of the top result. It is possible, however, to associate it to the push output and collapse both into $\mathbf{1}$, via $!_{P+\mathbf{1}}$. Thus, define

$$\mathsf{RightS} \;=\; \mathsf{Stack}[\iota_2 + \nabla, (\mathsf{id}+!_{P+\mathbf{1}}) \cdot \mathsf{a}_+] \;:\; \mathbf{1} + (P+P) \longrightarrow P + \mathbf{1}$$

Then, taking $\mathsf{LeftS} = \mathsf{Stack}$, we form the $\boxplus$ composition of both components:

$$\mathsf{RightS} \boxplus \mathsf{LeftS} :\; (\mathbf{1} + (P+P)) + (\mathbf{1} + \mathbf{1} + P) \longrightarrow (P + \mathbf{1}) + (P + P + \mathbf{1})$$

The next step builds the desirable connections using partial feedback over this composite. Supposing both the input and output type expressions are re-written in the additive left associative canonical form, in order to identify its components positionally, the intended push - pop connections are formed by feeding back the fourth output parcel (pop in LeftS) to the third input one (push in RightS) to model tr and, similarly, the first output parcel to the sixth input one to model tl. Formally, to be able to apply the *partial feedback* combinator, the two stacks composition has to be wrapped by a pair of suitable isomorphisms. In a diagram we have


$(\mathbf{1} + P + \mathbf{1} + \mathbf{1}) + (P + P)$
$(\mathsf{RightS} \boxplus \mathsf{LeftS})[\mathsf{wi}, \mathsf{wo}]$
$P + P$
$(\mathbf{1} + P + \mathbf{1}) + (P + P)$

(5.67)

and, formally,

$$\mathsf{AlmostFolder} \;=\; ((\mathsf{RightS} \boxplus \mathsf{LeftS})[\mathsf{wi}, \mathsf{wo}])\, {}^{\Lsh}{}_{P+P}$$

where

$$\begin{aligned} \mathsf{wi} &= \Big[\big[[[\iota_{11}, \iota_{121}], \iota_{112}], \iota_{212}\big], \big[\iota_{22}, \iota_{221}\big] \Big] \\ \mathsf{wo} &= \Big[[\iota_{12}, \iota_{111}], \big[[\iota_{211}, \iota_{22}], \iota_{21}\big] \Big] \end{aligned}$$

Finally, to conform $\mathsf{AlmostFolder}$ to the Folder interface, we will restrict the feed back input — by pre-composing with $\mathsf{fi} = \iota_1$ — and collapse both the trivial output and the feed back one to $\mathbf{1}$, by post-composing with $\mathsf{fo} = \big[[[\iota_2, \iota_1], \iota_2], \iota_2 \cdot !_{P+P}\big]$. Therefore, we complete the exercise by defining

$$\mathsf{Folder} \;=\; ((\mathsf{RightS} \boxplus \mathsf{LeftS})[\mathsf{wi}, \mathsf{wo}]) \Lsh_{P+P} \;\; [\mathsf{fi}, \mathsf{fo}]$$

which respects the intended interface.

64. NOTATION. The Folder example above unveils a notational problem in the straight application of the component algebra in practical situations. In particular, the use of *partial feedback* requires a huge amount of (isomorphic) wiring which degrades readability. The problem is, in fact, more general and a proper solution will probably require the introduction of a formal *diagrammatic* notation for composing components in which most wiring schemes are suitably encoded. Of course, the component algebra introduced earlier in this chapter will provide the semantics of such a notation. Although this thesis does not go further into this direction, a particular strategy to deal with feedbacks in a simpler, hopefully clearer, way is presented next.

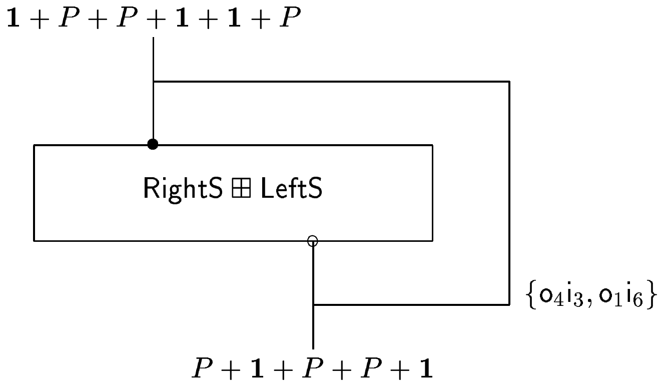
The idea is to introduce a *derived* feedback operator in which the connected interface points are explicitly referred — all the effectively required wiring is left implicit. A first approach could use the actual *action names* as arguments of the combinator. For example, the *folder* construction would be denoted as

$$\mathsf{Folder} \;=\; (\mathsf{RightS} \boxplus \mathsf{LeftS}) \Lsh_{\{\mathsf{L.pop}\!-\!\mathsf{R.push}, \mathsf{R.pop}\!-\!\mathsf{L.push}\}} \;\; [\cdots]$$

assuming actions on $\mathsf{RightS} \boxplus \mathsf{LeftS}$ have been renamed as indicated. Of course, the subsequent wiring to conform the Folder interface has to be written in terms of the $\mathsf{RightS} \boxplus \mathsf{LeftS}$ interface.

Such a solution, although notionally convenient in some situations, is not general enough for a number of reasons. First of all because often we want to feed back not the result of a particular action but just part of it or even only parcels of different outputs. In general, moreover, we refrain from associating interface types to explicitly named actions.

Positional specification appears to be a better solution. We assume that, prior to applying feedback, both the input and output interfaces are re-written into the additive left associative ('normal') form. If input and output parcels are numbered sequentially, from left to right, a connection between, say, output m and input n will be written as $\mathsf{o}_m\mathsf{i}_n$. Under this convention diagram (5.67), corresponding to AlmostFolder, would be replaced by


$\mathbf{1} + P + P + \mathbf{1} + \mathbf{1} + P$
RightS ⊞ LeftS
$\{\mathsf{o}_4\mathsf{i}_3, \mathsf{o}_1\mathsf{i}_6\}$
$P + \mathbf{1} + P + P + \mathbf{1}$

Such a notation is non ambiguous (at the cost of forcing explicit parcel aggregation if needed) and extends to the more general situation of *nested* feedback as discussed in the next paragraph.

65. INTERACTION SEQUENCES. Nested feedbacks occur wherever a given output is used as an input whose result is again supplied to the component. Such a process may continue revealing an *ordered* interaction pattern. Consider, for example, a component $p : W + Z \longrightarrow Z + W$ which interacts in, say, the following way: whenever a Z output is produced it is fed back to p and if, as a result, a W output is produced it is fed back as well. Such a component would be written as

$$(((p\,[\mathsf{id}, \mathsf{s}_+])\Lsh_Z)\,[\mathsf{s}_+, \mathsf{s}_+])\Lsh_W \tag{5.68}$$

This expression, as any other written in the component algebra notation, specifies the component's 'one-step' dynamics. Its (eventually infinite) behaviour, in which such interaction scheme repeats itself indefinitely, is only revealed by the component's anamorphic image.

It is easy to grasp that, in non trivial cases, the amount of wiring needed to specify nested feedbacks is rather large and a serious notational drawback. A possible simplification consists in extending the positional access convention discussed in §64 to cope with the interaction order. We assume the input and output indexes are always relative to the original interface (*i.e.*, before feedback application) re-written in the additive left associative normal form. The feedback expression, which decorates the derived feedback combinator, however, is no longer a set of input-output points —

which we call now a *plain interaction* — but a *sequence* of them. Formally, define

$$\begin{aligned} Interaction &= \mathcal{P}(OP \times IP) \\ ISeq &= Interaction^* \end{aligned}$$

where $OP \cong IP \cong \mathbb{N}$, subject to a data type invariant ensuring, for each $s \in Interaction$, the absence of repeated output (respectively, input) indexes. As a tentative convention, we decorate such indexes with, respectively, o and i. Plain interactions are written as sets and those are separated by a semi-colon in an interaction sequence. For example, expression (5.68) above would be simply written as

$$p \,{}^{\Lsh}\!\!\rceil_{\{\mathsf{o}_1\mathsf{i}_2\};\{\mathsf{o}_2\mathsf{i}_1\}}$$

We shall not proceed further with this topic which has to do more with the *pragmatics* of the component's algebra than with its *semantics*. We shall now discuss a special and rather common, class of components in which both feedback and restriction can be formulated in an alternative way.

5. Separable Components

66. Overview. Often a component is specified as a collection of actions over a shared state space, each of which exhibiting its own input and output types. Packing them together results in a component with an additive interface such that each type of input stimulus produces a result whose type is known and unique (among the possible results). Such components arise typically in the practical use of model oriented specification methods, such as VDM or Z. In particular, they admit combinators which restrict interfaces. In the next paragraphs a *restriction* combinator and a generalisation of the feed back interaction scheme, called *hook*, will be defined and some of their properties investigated.

67. Basic Definition. To be precise, let us call *separable* a component $p : I + J \longrightarrow O + R$ whose dynamics $\overline{a}_p$ can be split into two independent coalgebras

$$\overline{a}_{1.p} \;: U_p \longrightarrow \mathsf{B}(U_p \times O)^I$$

and

$$\overline{a}_{2.p} \;: U_p \longrightarrow \mathsf{B}(U_p \times R)^J$$

This notation suggests that $1.p$ and $2.p$ can also be regarded as independent components, with different interfaces, but defined over the same state space U_p and seed value u_p. Therefore the dynamics of p arises as the currying of

$$a_p \;=\; [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \mathsf{dr}$$

Clearly, for each behaviour monad B and objects I, J, O and R, separable components form a subcategory of the corresponding $\mathsf{Cp_B}$ hom category $\mathsf{Cp_B}(I+J, O+R)$. Moreover, Cp-arrows connecting separable components are characterised by the following property:

68. LEMMA. Let $p, q : I + J \longrightarrow O + R$ be separable. Then $h : p \longrightarrow q$ iff the same h, seen as an arrow in the underlying category, is also a comorphism from $1.p$ to $1.q$ and $2.p$ to $2.q$ (with a slight abuse of notation we shall write $h : 1.p \longrightarrow 1.q$ and $h : 2.p \longrightarrow 2.q$).

Proof. We first prove the right to left implication. Assume $h : 1.p \longrightarrow 1.q$ and $h : 2.p \longrightarrow 2.q$. Then

$$\begin{array}{cl}
& \mathsf{B}(h \times \mathsf{id}) \cdot a_p \\
= & \quad \{ \ p \text{ separable} \ \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ fusion, identity and } + \text{ absorption} \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p} + \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p}) \cdot \mathsf{dr} \\
= & \quad \{ \ \text{assumption: } h : 1.p \longrightarrow 1.q \text{ and } h : 2.p \longrightarrow 2.q \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.q} \cdot (h \times \mathsf{id}) + a_{2.q} \cdot (h \times \mathsf{id})) \cdot \mathsf{dr} \\
= & \quad \{ \ \mathsf{dr} \text{ natural} \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.q} + a_{2.q}) \cdot \mathsf{dr} \cdot (h \times \mathsf{id}) \\
= & \quad \{ \ q \text{ separable} \ \} \\
& a_q \cdot (h \times \mathsf{id})
\end{array}$$

For the reverse implication, assume $h : p \longrightarrow q$. Then,

$$\begin{array}{cl}
& h : p \longrightarrow q \\
\equiv & \quad \{ \ \text{comorphism condition and } p, q \text{ separable} \ \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.p} + a_{2.p}) \cdot \mathsf{dr} \\
& = \ [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.q} + a_{2.q}) \cdot \mathsf{dr} \cdot (h \times \mathsf{id}) \\
\equiv & \quad \{ \ + \text{ fusion, } + \text{ absorption and } \mathsf{dr} \text{ natural} \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p}] \cdot \mathsf{dr} \\
& = \ [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.q} \cdot (h \times \mathsf{id}), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.q} \cdot (h \times \mathsf{id})] \cdot \mathsf{dr} \\
\equiv & \quad \{ \ \text{equality} \ \} \\
& \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p} \ = \ \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.q} \cdot (h \times \mathsf{id})
\end{array}$$

$$
\begin{aligned}
& \text{and} \\
& \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p} \;=\; \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.q} \cdot (h \times \mathsf{id}) \\
= \quad & \qquad \{ \star \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p} \;=\; a_{1.q} \cdot (h \times \mathsf{id}) \\
& \text{and} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p} \;=\; a_{2.q} \cdot (h \times \mathsf{id}) \\
= \quad & \qquad \{ \text{ comorphism condition} \} \\
& h : 1.p \longrightarrow 1.q \;\text{ and }\; h : 2.p \longrightarrow 2.q
\end{aligned}
$$

where the $\star$ step is justified as follows. If $I \neq \emptyset$, ι_1 is a split mono and so is $\mathsf{B}(\mathsf{id} \times \iota_1)$, because any functor preserves split monos, and we are done. If $I = \emptyset$, the result holds trivially as both $\mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p}$ and $a_{1.q} \cdot (h \times \mathsf{id})$ have source $U_p \times \emptyset$ and any function from such a domain to an arbitrary set X can be written as $?_X \cdot \mathsf{zr}_{U_p}$. An similar argument establishes the second equation.

□

69. State Extension. Before analysing *restriction* and *hook*, the two extra combinators applicable to separable components, let us introduce the notion of *state extension* which will be required later and is generally useful in a number of situations. Let $p : I \longrightarrow 0$ be a component, V a set and $v \in V$. Then, define

$$
\begin{aligned}
p \otimes \langle v, V \rangle &= \langle \langle u_p, v \rangle \in U_p \times V, \overline{a}_{p \otimes \langle v,V \rangle} \rangle \\
\langle v, V \rangle \otimes p &= \langle \langle v, u_p \rangle \in V \times U_p, \overline{a}_{\langle v,V \rangle \otimes p} \rangle
\end{aligned}
$$

— where

$$
\begin{aligned}
a_{p \otimes \langle v,V \rangle} \;=\; & U_p \times V \times I \xrightarrow{\mathsf{xr}} U_p \times I \times V \xrightarrow{a_p \times \mathsf{id}} \mathsf{B}(U_p \times O) \times V \\
& \xrightarrow{\tau_r} \mathsf{B}(U_p \times O \times V) \xrightarrow{\mathsf{Bxr}} \mathsf{B}(U_p \times V \times O)
\end{aligned}
$$

and

$$
\begin{aligned}
a_{\langle v,V \rangle \otimes p} \;=\; & V \times U_p \times I \xrightarrow{\mathsf{a}} V \times (U_p \times I) \xrightarrow{\mathsf{id} \times a_p} V \times \mathsf{B}(U_p \times O) \\
& \xrightarrow{\tau_l} \mathsf{B}(V \times (U_p \times O)) \xrightarrow{\mathsf{Ba}^\circ} \mathsf{B}(V \times U_p \times O)
\end{aligned}
$$

— as the *right* or *left* state extension of p over V. Clearly, both constructions are functorial in $\mathsf{Cp}_\mathsf{B}(I, O)$, defining, for comorphism h, $h \otimes \langle v, V \rangle = h \times \mathsf{id}$ and $\langle v, V \rangle \otimes h = \mathsf{id} \times h$. Moreover, the behaviour of the extended component coincides with that of p itself, *i.e.*

$$
p \otimes \langle v, V \rangle \;\sim\; p \;\sim\; \langle v, V \rangle \otimes p \tag{5.69}
$$

for any $\langle v, V \rangle$.

Proof. We check that $\pi_1 \times \mathsf{id} : U_p \times V \longrightarrow U_p$ is a comorphism from $p \otimes \langle v, V \rangle$ to p. The left case is similarly verified.

$$\begin{array}{cl}
 & \mathsf{B}(\pi_1 \times \mathsf{id}) \cdot a_{p \otimes \langle v,V \rangle} \\
= & \qquad \{ \text{ definition } \} \\
 & \mathsf{B}(\pi_1 \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ routine: } (\pi_1 \times \mathsf{id}) \cdot \mathsf{xr} = \pi_1 \ \} \\
 & \mathsf{B}\pi_1 \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ law (C.12) } \} \\
 & \pi_1 \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \times \text{ cancellation } \} \\
 & a_p \cdot \pi_1 \cdot \mathsf{xr} \\
= & \qquad \{ \text{ routine: } (\pi_1 \times \mathsf{id}) = \pi_1 \cdot \mathsf{xr} \ \} \\
 & a_p \cdot (\pi_1 \times \mathsf{id})
\end{array}$$

□

70. Restriction. Let $\mathsf{Cp}_{\mathsf{B}}(I, O)_{Sp}$ denote the sub-category of $\mathsf{Cp}_{\mathsf{B}}(I, O)$ whose objects are *separable* components. Then, for each I, O, J and R, a *restriction* operator is defined as a family of functors ${}^{\blacktriangleright} : \mathsf{Cp}_{\mathsf{B}}(I + J, O + R)_{Sp} \longrightarrow \mathsf{Cp}_{\mathsf{B}}(I, O)_{Sp}$ which, being the identity on arrows, map each component $p : I + J \longrightarrow O + R$ to

$$ {}^{\blacktriangleright}p : I \longrightarrow O \ = \ \langle u_p \in U_p, \overline{a}_{1.p} \rangle \tag{5.70}$$

As ${}^{\blacktriangleright}h = h$ for any comorphism $h : p \longrightarrow q$, the functoriality conditions hold trivially. Also note that a similar (right) restriction operator is easily defined as

$$p^{\blacktriangleright} \ = \ {}^{\blacktriangleright}(p[\mathsf{s}_+, \mathsf{s}_+])$$

71. Lemma. Let $p : I + J \longrightarrow O + R$ separable. Then,

$$\ulcorner \iota_1 \urcorner ; p \ \sim \ {}^{\blacktriangleright}p ; \ulcorner \iota_1 \urcorner \tag{5.71}$$

Proof. Both components have type $I \longrightarrow O + R$. By law (5.10), $\ulcorner \iota_1 \urcorner \,;\, p \sim p[\iota_1, \mathsf{id}]$. Thus,

$$
\begin{aligned}
& a_{p[\iota_1,\mathsf{id}]} \\
= & \quad \{ \text{ wrapping definition and } p \text{ separable } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_1) \\
= & \quad \{ \text{ law (C.55) } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \iota_1 \\
= & \quad \{ \ + \text{ cancellation } \} \\
& \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p} \\
= & \quad \{ \text{ restriction definition } \} \\
& \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{\blacktriangleright p} \\
= & \quad \{ \text{ wrapping definition } \} \\
& a_{\blacktriangleright p[\mathsf{id},\iota_1]}
\end{aligned}
$$

which, again by (5.10), is bisimilar to $\blacktriangleright p \,;\, \ulcorner \iota_1 \urcorner$.

□

72. The result above formalises the intuition that restricting the input of a separable component also implies an output restriction, even if this is not directly expressed by the interface. For example, the output of $\ulcorner \iota_1 \urcorner \,;\, p$ is never of type R, even if the expression is typed $I \longrightarrow O + R$. The following lemma establishes under what conditions *separability* propagates under different forms of composition.

73. LEMMA. Let $p : I + J \longrightarrow O + R$, $q : O + R \longrightarrow P + T$, $r : Z + W \longrightarrow Z + W$ and $t : I + Z \longrightarrow O + Z$ be separable components. Then, $p \,;\, q$, $p[f + g, f' + g']$, $r \Lsh$ and $t \Lsh_Z$ are separable with

- $a_{1.(p;q)} = a_{1.p;1.q}$ and $a_{2.(p;q)} = a_{2.p;2.q}$
- $a_{1.(p[f+g,f'+g'])} = a_{1.p[f,f']}$ and $a_{2.(p[f+g,f'+g'])} = a_{2.p[g,g']}$
- $a_{1.(t \Lsh_Z)} = a_{1.t}$ and $a_{2.(t \Lsh_Z)} = a_{2.t} \bullet a_{2.t}$, where $\bullet$ denotes, as usual, the Kleisli composition for the behaviour monad B

Furthermore, let $p : I + J \longrightarrow O + R$ and $r : A \longrightarrow B$ be separable components. Then, $(p \boxplus r)[\mathsf{xr}_+, \mathsf{xr}_+]$, $(p \boxtimes r)[\mathsf{dl}^\circ, \mathsf{dl}]$ and $(p \boxast r)[\mathsf{wi}, \mathsf{wo}]$, where $\mathsf{wi} = (\mathsf{xr}_+ + \mathsf{dr}^\circ) \cdot \mathsf{m}_+$ and $\mathsf{wo} = \mathsf{m}_+ \cdot (\mathsf{xr}_+ + \mathsf{dr})$, are separable with

- $a_{1.((p \boxplus r)[\mathsf{xr}_+,\mathsf{xr}_+])} = a_{1.p \boxplus r}$ and $a_{2.((p \boxplus r)[\mathsf{xr}_+,\mathsf{xr}_+])} = a_{2.p \otimes \langle u_r, U_r \rangle}$
- $a_{1.((p \boxtimes r)[\mathsf{dl}^\circ,\mathsf{dl}])} = a_{1.p \boxtimes r}$ and $a_{2.((p \boxtimes r)[\mathsf{dl}^\circ,\mathsf{dl}])} = a_{2.p \boxtimes r}$
- $a_{1.((p \boxast r)[\mathsf{wi},\mathsf{wo}])} = a_{1.p \boxast r}$ and $a_{2.((p \boxast r)[\mathsf{wi},\mathsf{wo}])} = a_{2.p \boxast r}$

Proof. Consider first the case of sequential composition. The proof proceeds by showing that $p\,;\,q$ can be re-written as

$$a_{p;q} \;=\; [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.(p;q)}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.(p;q)}] \cdot \mathsf{dr}$$

where $a_{1.(p;q)}$ and $a_{2.(p;q)}$ are replaced by the conjecture given. A simple, although lengthy, calculation, will establish the result. The other cases are immediate. For the second group of compositions the proof argument is similar. Note, however, the need for introducing appropriate wiring to conform the result of the composition to the interface scheme of separable components. For example, the type of $p \boxast r$ is $(I+J+A)+((I+J)\times A) \longrightarrow (O+R+B)+((O+R)\times B)$ whereas, to conform to definition in §67, type $(I+A+I\times A)+(J+J\times A) \longrightarrow (O+B+O\times B)+(R+R\times B)$ is required. Proof details are given in [Appendix D, page 390].

□

74. Restriction Laws. The following laws, relating restriction to further operators of the component's algebra, are a direct corollary of lemma §73. Notice that, once separability propagation has been proved, the definition of restriction applies normally. Thus, let $p : I + J \longrightarrow O + R$, $q : O + R \longrightarrow P + T$ be separable and $r : A \longrightarrow B$ any component. Then,

$$\begin{aligned}
{}^{\blacktriangleright}p[f, f'] &\sim {}^{\blacktriangleright}(p[f+g, f'+g']) && (5.72)\\
{}^{\blacktriangleright}p\,;\,{}^{\blacktriangleright}q &\sim {}^{\blacktriangleright}(p\,;\,q) && (5.73)\\
{}^{\blacktriangleright}p \boxplus r &\sim {}^{\blacktriangleright}((p \boxplus r)[\mathsf{xr}_+, \mathsf{xr}_+]) && (5.74)\\
{}^{\blacktriangleright}p \boxtimes r &\sim {}^{\blacktriangleright}((p \boxtimes r)[\mathsf{dl}^\circ, \mathsf{dl}]) && (5.75)\\
{}^{\blacktriangleright}p \boxast r &\sim {}^{\blacktriangleright}((p \boxast r)[(\mathsf{xr}_+ + \mathsf{dr}^\circ) \cdot \mathsf{m}_+, \mathsf{m}_+ \cdot (\mathsf{xr}_+ + \mathsf{dr})]) && (5.76)
\end{aligned}$$

Furthermore, for $p : I + Z \longrightarrow I + Z$ and $q : I + Z \longrightarrow O + Z$ separable,

$$\begin{aligned}
({}^{\blacktriangleright}p) \Lsh &\sim {}^{\blacktriangleright}p \Lsh && (5.77)\\
({}^{\blacktriangleright}q) \Lsh_Z &\sim {}^{\blacktriangleright}q \Lsh_Z && (5.78)
\end{aligned}$$

75. Feedback Revisited. Let $p : I + Z \longrightarrow Z + O$ be a separable component. Then, a Z output fed back will always produce a O one. Thus, a new feedback combinator, which actually restricts the component interface, may be defined as

$$\big(p\big)_Z \; : I \longrightarrow O \;=\; \langle u_p \in U_p, \overline{a}_{\left(p\right)_Z} \rangle$$

where

$$a_{\left(p\right)_Z} \;=\; U_p \times I \xrightarrow{a_{2.p} \bullet a_{1.p}} \mathsf{B}(U_p \times O)$$

76. EXAMPLE. A particular case of a separable component is $q \boxplus r : I + Z \longrightarrow Z + O$, where $q : I \longrightarrow Z$ and $r : Z \longrightarrow O$. Clearly,

$$\left(q \boxplus r\right)_Z \;\sim\; q \,;\, r \tag{5.79}$$

Proof. As a corollary of lemma §73, $q \boxplus r$ is separable with $a_{1.(q \boxplus r)} = a_{q \otimes \langle u_r, U_r \rangle}$ and $a_{2.(q \boxplus r)} = a_{\langle u_q, U_q \rangle \otimes r}$. Note that, by law (5.69), $q \otimes \langle u_r, U_r \rangle \sim q$ and, similarly, $r \otimes \langle u_q, U_q \rangle \sim r$. Then,

$$
\begin{array}{ll}
& a_{\left(q \boxplus r\right)_Z} \\
= & \qquad \{ \textit{ hook} \text{ definition } \} \\
& \mu \cdot \mathsf{B} a_{\langle u_q, U_q \rangle \otimes r} \cdot a_{q \otimes \langle u_r, U_r \rangle} \\
= & \qquad \{ \textit{ state extension} \text{ definition } \} \\
& \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_r) \cdot \mathsf{Ba} \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_q \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ ; definition } \} \\
& a_{q;r}
\end{array}
$$

□

77. LEMMA. Let $p : I + Z \longrightarrow Z + O$ be a separable component. Then,

$$\left(p\right)_Z \,;\, \ulcorner \iota_1 \urcorner \;\sim\; \ulcorner \iota_1 \urcorner \,;\, (p[\mathsf{id}, \mathsf{s}_+]) \Lsh_Z \tag{5.80}$$

Proof. This result plays a role similar to lemma §71, in the sense that it also formalises an intuition, this time about feeding back into separable components: the feed back factor could be dropped from the interface. Proof supplied in [Appendix D, page 394].

□

78. THE HOOK COMBINATOR. A more general situation arises wherever p has type

$$p : I + J + Z \longrightarrow Z + O + R$$

and is separable into three 'threads' $1.p$, $2.p$ and $3.p$, with $1.p$ and $3.p$ composable (actually, according to definition §67, two 'threads' $(1.p + 2.p)$ and $3.p$, the first one expressed as a sum). Then, for each I, Z, J, R and O, the *hook* combinator is defined as a family of functors $(p)_Z : \mathsf{Cp}_{\mathsf{B}}(I{+}J{+}Z, Z{+}O{+}R)_{Sp} \longrightarrow \mathsf{Cp}_{\mathsf{B}}(I{+}J, R{+}O)_{Sp}$ which, being the identity on arrows, map each component $p : I{+}J{+}Z \longrightarrow Z{+}O{+}R$ to

$$(p)_Z : I + J \longrightarrow O + R \;=\; \langle u_p \in U_p, \overline{a}_{(p)_Z} \rangle$$

where

$$\begin{aligned} a_{(p)_Z} \;=\; & U_p \times (I + J) \xrightarrow{\qquad \mathsf{dr} \qquad} U_p \times I + U_p \times J \\ & \xrightarrow{(a_{3.p} \bullet a_{1.p}) + a_{2.p}} \mathsf{B}(U_p \times R) + \mathsf{B}(U_p \times O) \\ & \xrightarrow{[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]} \mathsf{B}(U_p \times (R + O)) \end{aligned}$$

Note that this definition subsumes the one in §75: by isomorphic wiring the interface of $p : I + Z \longrightarrow Z + O$ can be re-written as $I + \emptyset + O \longrightarrow Z + \emptyset + O$ which is separable, with the original p threads in the first and third positions and $?_{\mathsf{B}(U_p \times \emptyset)}$ in the second.

79. LEMMA. For each I, Z, J, R and O, the *hook* combinator defined above is a functor from $\mathsf{Cp}_{\mathsf{B}}(I + J + Z, Z + O + R)_{Sp}$ to $\mathsf{Cp}_{\mathsf{B}}(I + J, R + O)_{Sp}$.

Proof. The basic proof step will show that a comorphism $h : p \longrightarrow q$ is still a comorphism $h : (p)_Z \longrightarrow (q)_Z$. Then the functoriality conditions hold trivially. Thus,

$$\begin{aligned} & \mathsf{B}(h \times \mathsf{id}) \cdot a_{(p)_Z} \\ = & \qquad \{ \textit{ hook } \text{definition} \} \\ & \mathsf{B}(h \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot ((a_{3.p} \bullet a_{1.p}) + a_{2.p}) \cdot \mathsf{dr} \\ = & \qquad \{ + \text{ fusion and absorption, } \bullet \text{ definition} \} \\ & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(h \times \mathsf{id}) \cdot \mu \cdot \mathsf{B}a_{3.p} \cdot a_{1.p} + \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p}) \cdot \mathsf{dr} \\ = & \qquad \{ \mu \text{ natural (C.16)} \} \\ & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu \cdot \mathsf{BB}(h \times \mathsf{id}) \cdot \mathsf{B}a_{3.p} \cdot a_{1.p} + \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p}) \cdot \mathsf{dr} \\ = & \qquad \{ h : 2.p \longrightarrow 2.q \text{ and } h : 3.p \longrightarrow 3.q \text{ by lemma §68} \} \\ & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu \cdot \mathsf{B}a_{3.q} \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p} + a_{2.q} \cdot (h \times \mathsf{id})) \cdot \mathsf{dr} \\ = & \qquad \{ h : 1.p \longrightarrow 1.q \text{ again by lemma §68} \} \\ & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu \cdot \mathsf{B}a_{3.q} \cdot a_{1.q} \cdot (h \times \mathsf{id}) + a_{2.q} \cdot (h \times \mathsf{id})) \cdot \mathsf{dr} \end{aligned}$$

$$
\begin{array}{ll}
= & \{\ \mathsf{dr}\ \text{natural}\ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu \cdot \mathsf{B}a_{3.q} \cdot a_{1.q} + a_{2.q}) \cdot \mathsf{dr} \cdot (h \times \mathsf{id}) \\
= & \{\ \bullet\ \text{definition}\ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{3.q} \bullet a_{1.q} + a_{2.q}) \cdot \mathsf{dr} \cdot (h \times \mathsf{id}) \\
= & \{\ \textit{hook}\ \text{definition}\ \} \\
& a_{(q)_Z} \cdot (h \times \mathsf{id})
\end{array}
$$

□

80. Hook Laws. We shall conclude our digression around separable components by stating some properties of the interaction between *hook* and other component combinators. Proof arguments rely on the separability of the composites, now into three 'threads' (see lemma §73). We stress the intuition about each law, omitting lengthy, but technically simple, calculational details.

The first two laws relate *hook* with wrapping and restriction. Note that *hook* is well behaved with respect to a 'structural' wrapping function, provided its component in the feed back parameter is an isomorphism. Thus, let $p : I + J + Z \longrightarrow Z + O + R$ be a separable component and $i : W \longrightarrow Z$ an isomorphism. Then,

$$(p[f + g + i, i^{\circ} + h + k])_W \ \sim\ (p)_Z[f + g, k + h] \tag{5.81}$$

$$(\,{}^{\blacktriangleright}p[\mathsf{xr}_+, \mathsf{xr}_+])_Z \ \sim\ {}^{\blacktriangleright}(p)_Z \tag{5.82}$$

Proof. As an illustration of the kind of arguments involved, we prove (5.81) below.

$$
\begin{array}{ll}
& a_{(p[f+g+i, i^{\circ}+h+k])_{Z'}} \\
= & \{\ \textit{hook}\ \text{definition}\ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{3.p[f+g+i,i^{\circ}+h+k]} \bullet a_{1.p[f+g+i,i^{\circ}+h+k]} \\
& +a_{2.p[f+g+i,i^{\circ}+h+k]}) \cdot \mathsf{dr} \\
= & \{\ \text{'threads' of a wrapped component —see §73}\ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot ((\mathsf{B}(\mathsf{id} \times k) \cdot a_{3.p} \cdot (\mathsf{id} \times i)) \bullet (\mathsf{B}(\mathsf{id} \times i^{\circ}) \cdot a_{1.p} \cdot (\mathsf{id} \times f)) \\
& +\mathsf{B}(\mathsf{id} \times h) \cdot a_{2.p} \cdot (\mathsf{id} \times g)) \cdot \mathsf{dr} \\
= & \{\ \bullet\ \text{definition}\ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu \cdot \mathsf{B}(\mathsf{B}(\mathsf{id} \times k) \cdot a_{3.p} \cdot (\mathsf{id} \times i)) \cdot \mathsf{B}(\mathsf{id} \times i^{\circ}) \cdot a_{1.p} \cdot (\mathsf{id} \times f) \\
& +\mathsf{B}(\mathsf{id} \times h) \cdot a_{2.p} \cdot (\mathsf{id} \times g)) \cdot \mathsf{dr}
\end{array}
$$

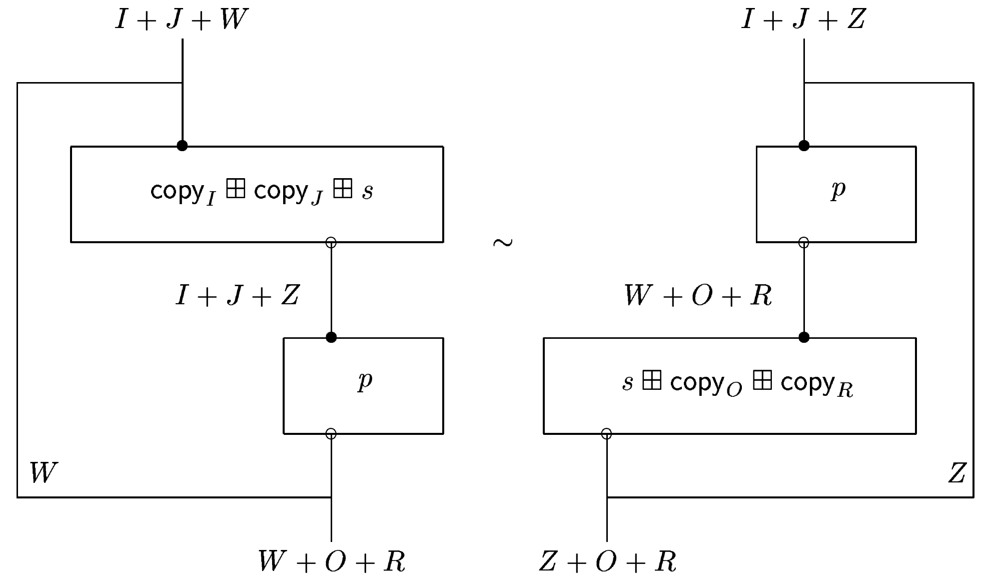
$$
\begin{array}{ll}
= & \quad \{ \ i \text{ isomorphism} \ \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu \cdot \mathsf{BB}(\mathsf{id} \times k) \cdot \mathsf{B}a_{3.p} \cdot a_{1.p} \cdot (\mathsf{id} \times f) \\
 & +\mathsf{B}(\mathsf{id} \times h) \cdot a_{2.p} \cdot (\mathsf{id} \times g)) \cdot \mathsf{dr} \\
= & \quad \{ \ \mu \text{ natural (C.16) and functors} \ \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{id} \times k) + \mathsf{B}(\mathsf{id} \times h)) \cdot (\mu \cdot \mathsf{B}a_{3.p} \cdot a_{1.p} + a_{2.p}) \\
 & \cdot(\mathsf{id} \times f + \mathsf{id} \times g) \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ absorption and fusion, } \mathsf{dr} \text{ natural} \ \} \\
 & \mathsf{B}(\mathsf{id} \times (h+k)) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu \cdot \mathsf{B}a_{3.p} \cdot a_{1.p} + a_{2.p}) \cdot \mathsf{dr} \cdot (\mathsf{id} \times (f+g)) \\
= & \quad \{ \ \textit{hook} \text{ and } \textit{wrapping} \text{ definitions} \ \} \\
 & a_{(p)_Z[f+g,k+h]}
\end{array}
$$

□

The *hook* combinator can be thought of as a *partial* sequential composition, in the sense that joined ‘pins’ vanish from the outermost interface. This gives rise to a number of bisimilar aggregation schemes, based on either **;** or ⊞, allowing the specifier to play around with the components involved. For example, the ‘*hook* version’ of (5.66) reads

$$\big((p \boxplus r)\,[\mathsf{xr}_+, \mathsf{id}]\big)_Z \ \sim \ (p)_Z \boxplus r \tag{5.83}$$

for $p : I + Z \longrightarrow Z + O$ and $r : J \longrightarrow R$. On the other hand, a number of laws make it possible to swap a partial composition from the input to the output of the feed back ‘pin’. The two cases mentioned below are prototypical of this class of situations — respectively, in **;** and ⊞ composition contexts. Let $p : I + J + Z \longrightarrow W + O + R$ be separable and $s : W \longrightarrow Z$. Then,


$I+J+W$
$\mathsf{copy}_I \boxplus \mathsf{copy}_J \boxplus s$
$I+J+Z$
p
W
$W+O+R$
$\sim$
$I+J+Z$
p
$W+O+R$
$s \boxplus \mathsf{copy}_O \boxplus \mathsf{copy}_R$
Z
$Z+O+R$

Formally,

$$\big((\mathsf{copy}_I \boxplus \mathsf{copy}_J \boxplus s) \,;\, p\big)_W \;\sim\; \big(p \,;\, (s \boxplus \mathsf{copy}_O \boxplus \mathsf{copy}_R)\big)_Z \tag{5.84}$$

For a similar result, in a $\boxplus$ context, let $p : I + Z \longrightarrow O + R$, $r : J \longrightarrow W$ and $s : W \longrightarrow Z$. Then,

$$\big(((r \,;\, s) \boxplus p)\,[\mathsf{a}_+, \mathsf{a}_+]\big)_Z \;\sim\; \big((r \boxplus ((\mathsf{copy}_I \boxplus s) \,;\, p))\,[\mathsf{a}_+, \mathsf{a}_+]\big)_W \tag{5.85}$$

The proof idea for (5.84) is that, because p is separable, component s is activated in both expressions only once and always triggered by a p response to an I typed input. A lengthy, but trivial, expression manipulation establishes $\mathsf{s} \cdot (\mathsf{s} \times \mathsf{id}) : \mathbf{1} \times \mathbf{1} \times U_s \times U_p \longrightarrow U_p \times (U_s \times \mathbf{1} \times \mathbf{1})$ as a comorphism from the left to the right hand side of the equation. A similar argument establishes (5.85).

81. ELECTRONIC VOTING. Our last example illustrates the role of separability. Let us consider a voting system in which stimuli sent by independent voting pads are counted in a central unit ('concentrator') until a certain level is reached. A common use of such a system can be found in processing electronic opinion polls, as in some television shows. In this case the voting pad is used only once. However, the same system can be used to count inputs from a number of sensors in, *e.g.*, an industrial plant. Typically, such sensors emit a number of stimuli before terminating. In any case, the *maybe* monad seems an adequate choice for the behaviour model. The voting system is built around two basic components: the *voting pad* VP and the *concentrator* C specified as follows:

$$\mathsf{VP} : \mathbf{1} \longrightarrow \mathbf{1} \quad = \quad \langle n \in \mathbb{N}, \overline{a}_{\mathsf{VP}} \rangle$$

where

$$a_{\mathsf{VP}}\ \langle n, * \rangle\ = (n \neq 0\ \rightarrow\ \iota_1\ \langle n-1, * \rangle,\ \iota_2\ *)$$

and

$$\mathsf{C} : \mathbb{N} + \mathbf{1} \longrightarrow \mathbf{1} + \mathbf{2} \quad = \quad \langle 0 \in \mathbb{N}, \overline{a}_{\mathsf{C}} \rangle$$

whose dynamics $\overline{a}_{\mathsf{C}}$ is based on two actions: reset, to set the minimum vote level needed to report success, and vote to count an individual vote. Formally,

$$\begin{aligned} \mathsf{reset}\ \langle u, m \rangle\ &= \iota_1\ \langle m, * \rangle \\ \mathsf{vote}\ \langle u, * \rangle\ &= \iota_1\ \langle u-1, u' = 1 \rangle \end{aligned}$$

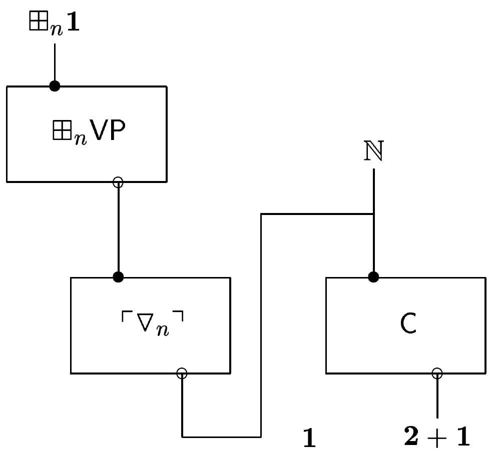
A n-voting system is assembled by aggregating n voting pads and connecting their outputs to the concentrator. A n-codiagonal wire is needed to concentrate the voting pads' outputs. As C is separable, the *hook* combinator can be used for interaction. Thus, we begin with

$$\mathsf{S}_n \ = \ (\boxplus_n \mathsf{VP}\ ;\ulcorner \nabla_n \urcorner) \boxplus \mathsf{C}$$

which is typed as $\boxplus_n \mathbf{1} + (\mathbb{N} + \mathbf{1}) \longrightarrow \mathbf{1} + (\mathbf{1} + \mathbf{2})$. To apply *hook*, however, S has to be wired to exhibit the hooked type in the correct position. Clearly, $\mathsf{S}[\mathsf{a}_+, \mathsf{a}_+]$ has the right type: $\boxplus_n \mathbf{1} + \mathbb{N} + \mathbf{1} \longrightarrow \mathbf{1} + \mathbf{1} + \mathbf{2}$. The voting system is, then, defined as

$$\mathsf{VS}_n \ = \ \big(((\boxplus_n \mathsf{VP}\ ;\ulcorner \nabla_n \urcorner) \boxplus \mathsf{C})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbf{1}} \quad : \quad \boxplus_n \mathbf{1} + \mathbb{N} \longrightarrow \mathbf{2} + \mathbf{1}$$

which may be depicted as



82. Concurrent Votes. In VS_n each vote is dealt separately. Replacing $\boxplus$ by $\boxast$ as the 'gluing' combinator of the voting pads, allows for the simultaneously counting of arbitrary chunks of votes. Eventually this suits reality better, as several voting pads may be activated at the same time. Two extra modifications are required in the system to cope with this new specification. First, the vote button of the concentrator has to be re-designed to accept, instead of a single stimulus, a natural number encoding the number of incoming votes, ie,

$$\mathsf{vote}\ \langle u, n\rangle \ = \iota_1\ \langle u - n, u' \leq 0\rangle$$

Together with the previous reset button, this defines a new concentrator $\mathsf{NC} : \mathbb{N} + \mathbb{N} \longrightarrow \mathbf{1} + \mathbf{2}$. Secondly, the codiagonal wiring has to be replaced by a function $\mathsf{count}_n : \boxast_n \mathbf{1} \longrightarrow \mathbb{N}$ which actually counts the number of inputs received. For $n = 2$, $\mathsf{count}_2 : (\mathbf{1} + \mathbf{1}) + \mathbf{1} \times \mathbf{1} \longrightarrow \mathbb{N}$ would simply be $[\underline{1}\cdot !, \underline{2}\cdot !]$. The new system is, then, assembled as in the VS_n case:

$$\mathsf{CVS}_n \ = \ \big(((\boxast_n \mathsf{VP} \,;\, \ulcorner \mathsf{count}_n \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \quad : \quad \boxast_n \mathbf{1} + \mathbb{N} \longrightarrow \mathbf{2} + \mathbf{1}$$

Clearly, CVS_n exhibits a 'richer' behavioural pattern than VS_n, in a very precise sense: if input to CVS_n is restricted to a sum of stimuli, the resulting component becomes bisimilar to VS_n. Let us prove this for $n = 2$ (for $n > 2$ the proof requires some extra wiring manipulation).

Proof.

$$\begin{array}{cl}
 & \ulcorner \iota_1 + \mathsf{id} \urcorner \,;\, \mathsf{CVS}_2 \\
\sim & \qquad \{\ \mathsf{CVS}_2 \text{ definition and law (5.10)} \ \} \\
 & \big(((\mathsf{VP} \boxast \mathsf{VP} \,;\, \ulcorner \mathsf{count}_n \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}}\,[\iota_1 + \mathsf{id}, \mathsf{id}] \\
\sim & \qquad \{\ \text{law (5.81)} \ \} \\
 & \big((((\mathsf{VP} \boxast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_n \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+][\iota_1 + \mathsf{id} + \mathsf{id}, \mathsf{id}]\big)_{\mathbb{N}} \\
\sim & \qquad \{\ \mathsf{a}_+ \text{ natural and law (5.10)} \ \} \\
 & \big((\ulcorner \iota_1 + \mathsf{id} \urcorner \,;\, ((\mathsf{VP} \boxast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \qquad \{\ \text{law (5.17) and } \mathsf{copy}_X \text{ definition} \ \} \\
 & \big(((\ulcorner \iota_1 \urcorner \boxplus \mathsf{copy}_N) \,;\, ((\mathsf{VP} \boxast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \qquad \{\ \text{law (5.15)} \ \} \\
 & \big(((\ulcorner \iota_1 \urcorner \,;\, ((\mathsf{VP} \boxast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner)) \boxplus (\mathsf{copy}_N \,;\, \mathsf{NC}))\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \qquad \{\ \text{laws (5.4) and (5.5)} \ \} \\
 & \big(((\ulcorner \iota_1 \urcorner \,;\, (\mathsf{VP} \boxast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}}
\end{array}$$

$$
\begin{array}{ll}
\sim & \quad \{ \text{ laws (5.53) } \} \\
& ((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, \ulcorner \iota_1 \urcorner \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\, [\mathsf{a}_+, \mathsf{a}_+])_{\mathbb{N}} \\
\sim & \quad \{ \text{ laws (5.5) } \} \\
& ((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, (\ulcorner \iota_1 \urcorner \,;\, \ulcorner \mathsf{count}_2 \urcorner)) \boxplus \mathsf{NC})\, [\mathsf{a}_+, \mathsf{a}_+])_{\mathbb{N}} \\
\sim & \quad \{ \ \mathsf{count}_2 \cdot \iota_1 = \underline{1} \cdot !_{1+1} = \underline{1} \cdot \nabla \ \} \\
& ((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, (\ulcorner \nabla \urcorner \,;\, \ulcorner \underline{1} \urcorner)) \boxplus \mathsf{NC})\, [\mathsf{a}_+, \mathsf{a}_+])_{\mathbb{N}} \\
\sim & \quad \{ \text{ law (5.85) } \} \\
& ((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, \ulcorner \nabla \urcorner) \boxplus ((\mathsf{copy}_{\mathbb{N}} \boxplus \ulcorner \underline{1} \urcorner) \,;\, \mathsf{NC}))\, [\mathsf{a}_+, \mathsf{a}_+])_{\mathbf{1}} \\
\sim & \quad \{ \ (\mathsf{copy}_{\mathbb{N}} \boxplus \ulcorner \underline{1} \urcorner) \,;\, \mathsf{NC} \sim \mathsf{C} \ \} \\
& ((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, \ulcorner \nabla \urcorner) \boxplus \mathsf{C})\, [\mathsf{a}_+, \mathsf{a}_+])_{\mathbf{1}} \\
\sim & \quad \{ \ \mathsf{VS}_2 \text{ definition } \} \\
& \mathsf{VS}_2
\end{array}
$$

□

6. Animating Bh_{B}

83. In chapter 4 we have seen in some detail how CHARITY provides a computational representation of processes on top of which different process calculi can be prototyped. In a similar way, CHARITY can be used to ‘bring life’ to the category of components’ behaviours, Bh_{B}. Again the prototyping process relies on the explicit support CHARITY provides for coinductive types, as explained in appendix E.

This section discusses a CHARITY implementation of the component algebra introduced in this chapter. The implementation is parametric on the underlying behaviour model. Associated to each such model, a specific *observation structure* is proposed and a correspondent testing function defined. The reader is referred to appendix E for a basic introduction to the CHARITY notation.

84. DETERMINISTIC COMPONENTS. We shall consider first the class of *deterministic* components, *i.e.*, the case in which $\mathsf{B} = \mathsf{Id}$. This is, of course, a particular case of the generic implementation scheme presented later, but it seems a convenient starting point to explain the basic prototyping strategies. This type of deterministic behaviours is declared in CHARITY as

```
data U -> cp(I,O) =  a : U -> I => U * O.
```

parametric on the input (I) and output (O) types. As an example consider cell which, given an integer seed value, generates the behaviour of a component which reacts to an integer stimulus by doubling the value of the state variable and producing as output the sum of the current state value with the input received. Notice how the component semantics is given as the ‘gene’ of an anamorphim:

```
def cell: int -> cp(int,int)
    = u => (| u => a: n => (double u, add_int(u,n)) |) u.
```

A small variation of cell is conv, in which the output commutes from (to) integers to (from) naturals (functions n2i and i2n make the obvious conversions):

```
def conv: int -> cp(int+nat,int+nat)
    = u => (| u => a: b0 i => (double u, b1 i2n add_int(u,i))
                    | b1 n => (double u, b0 add_int(u, n2i n))
            |) u.
```

85. COMBINATORS. Behaviour combinators are implemented in CHARITY as anamorphisms whose ‘genes’ are direct translations of the corresponding operators in Cp. Despite of making use of the CHARITY term logic to handle variables and some expression ‘housekeeping’, the correspondence with the definitions in sections 1 to 3, bearing in mind that $\mathsf{B} = \mathsf{Id}$, should be obvious. Let us see some implementations. The *sequential composition* and its unit are given by

```
def pipe: cp(I, K) * cp(K, O) -> cp(I, O)
    = pp => (| (p, q) => a: i => a(i,p)
                            ; (p',k) => a(k,q)
                            ; (q',o) => ((p', q'), o)
             |) pp.
```

and

```
def copy: 1 -> cp(K,K)
    = () => (| () => a: k => ((), k) |) ().
```

The code for copy, which has been defined as the lifting of id_K, gives the general pattern for function lifting. Thus,

```
def lift{f:I -> O}: 1 -> cp(I,O)
    = () => (| _ => a: i => ((), f i ) |) ().
```

Notice lift is parametrized by the function f to be lifted to Cp. A similar parametrization appears in the following implementation of *wrapping*:

```
def wrap{f:I' -> I, g:O -> O'}: cp(I, O) -> cp(I',O')
    = pp => (| p => a: i' => a(f i', p)
                     ; (p',o) => (p', g o)
             |) pp.
```

On their turn, the *choice* and *parallel* combinators on behaviours are implemented as

```
def par: cp(I, O) * cp(J, R) -> cp(I*J, O*R)
    = pp => (| (p, q) => a: (i,j) => (a(i,p), a(j,q))
                           ; ((p',o),(q',r)) => ((p', q'), (o,r))
             |) pp.
```

and

```
def choice: cp(I, O) * cp(J, R) -> cp(I+J, O+R)
    = pp =>
     (| (p, q) => a: b0 i => a(i,p) ; (p',o) => ((p', q), b0 o)
                   | b1 j => a(j,q) ; (q',r) => ((p, q'), b1 r)
      |) pp.
```

Derived combinators such as, for example, *either*, have direct definitions, as in

```
def either: cp(I, O) * cp(J, O) -> cp(I+J, O)
    =  (p,q) => pipe(choice(p,q), lift{codiag}()).
```

Finally, the definitions of *feedback* and *partial feedback* go as follows:

```
def feed: cp(K, K) -> cp(K,K)
    = pp => (| p => a: k => a(k,p) ; (p',k') => a(k',p') |) pp.

def pfeed: cp(I+K, O+K) -> cp(I+K, O+K)
    = pp =>
    (| p => a: x => a(x,p)
         ; (p', y) => {b0 o => (p',b0 o) | b1 k => a(b1 k,p')} y
     |) pp.
```

The application of the `a` observer to a particular stimulus and a behaviour produces, as expected, an output value and the ‘continuation’ behaviour. Four examples of these are given below in CHARITY printouts, showing, respectively, the execution of `cell` with 5 as a seed value, the sequential composition of two such components with different seeds, feedback and partial feedback applied, respectively, to `cell` and `conv`.

```
>> a(4, cell 5).
((a: <function>), 9) : cp(int, int) * int

>> a(4, pipe (cell 5, cell 2)).
((a: <function>), 11) : cp(int, int) * int

>> a(4, feed cell 5).
((a: <function>), 19) : cp(int, int) * int

>> a(b0 2, pfeed conv 2).
((a: <function>), b0(8)) : cp((int+nat),(int+nat)) * (int+nat)
```

Notice, in the last example, that input `b0 2` produces a `nat` result, which is fed back and, finally, produces the result shown. As a last example, consider the parallel execution of two `cell` behaviours but in a way such that the interface is changed to type `nat`. Function `fp`, defined in §88 below, computes the product of two functions.

```
>> a((two,one), wrap{fp{n2i,n2i}, fp{i2n,i2n}} par (cell 1, cell 0)).
  ((a: <function>), (succ(succ(succ(zero))), succ(zero)))
   : cp((nat * nat), (nat * nat)) * (nat * nat)
```

86. TESTING. We have just seen the sort of interaction CHARITY provides. As the elements of the final coalgebra associated to the signature functor for components are themselves functions, the user is required to supply further input if experimentation is intended to proceed. Behaviour is revealed step by step along this process. For example, the reaction of `cell 1` to, say, input 5, after having reacted to input 8, is computed by

```
>> a(5, p0 a(8, cell 1)).
((a: <function>), 7) : cp(int, int) * int
```

If enough time is given, a deterministic component $p : I \longrightarrow O$ produces a stream of O-values as a reaction to a stream of I stimuli. Actually, streams are the appropriate *observation structure* for deterministic behaviours. This may be used to devise a testing function which populates a result stream by successively experimenting the prototype with a supplied input stream. The CHARITY mechanism to (lazily) evaluate coinductive types provides a practical way to implement this.

Recall that streams of O are elements of the carrier of the final coalgebra for $\mathsf{T} = O \times \mathsf{Id}$ (§E.3). Thus, the testing function `obs` arises as the anamorphism depicted in the following diagram:

$$\begin{tikzcd}
\texttt{stream O} \arrow[r, "\omega_{\mathsf{O}\times\mathsf{Id}}"] & \texttt{O} \times \texttt{stream O} \\
\texttt{stream I} \times \texttt{cp(I,O)} \arrow[u, "\texttt{obs}=[\!(\texttt{obs\_gen})\!]"] \arrow[r, "\texttt{obs\_gen}"] & \texttt{O} \times (\texttt{stream I} \times \texttt{cp(I,O)}) \arrow[u, "\mathsf{id}\times\texttt{obs}"']
\end{tikzcd}$$

that is,

```
def obs: stream I * cp(I,O) -> stream O
    = (s,p)  => (| (s,p) => head: p1 a(head s, p)
                |          tail: (tail s, p0 a(head s,p))
                |) (s, p).
```

where `obs_gen` is the 'gene' coalgebra specified between `(|` and `|)`.
Observe that `obs_gen` returns a pair formed by the component output, produced as a reaction to the head of the input stream, and another pair formed by the tail of the input stream and the 'continuation' behaviour. As an example, consider the response of `cell` to `ints`, the stream of integers 1, 2, $\cdots$ (stream generation is discussed in §E.4):

```
>> obs(ints, cell 4).
(head: ..., tail: ...)

Right display mode:
(q - quit, return - more) >>
(head: 4, tail: (head: ..., tail: ...))

Right display mode:
(q - quit, return - more) >>
(head: 4, tail: (head: 9, tail: (head: ..., tail: ...)))

Right display mode:
(q - quit, return - more) >>
(head: 4, tail: (head: 9, tail: (head: 18, tail: ...)))

Right display mode:
(q - quit, return - more) >>
(head: 4, tail: (head: 9, tail: (head: 18, tail: (head: 35, ...))))
...
```

However, testing the feedback combinators is rather unpleasant: the 'continuation' behaviour is returned as a function which the user must animate with the output produced, if interested in the 'long term' feed back behaviour. In fact, the reaction

```
((a: <function>), 10) : cp(int, int) * int
```

to, `a(4, feed cell 2)` is produced in one step feedback loop and the prototyping system does not provide an immediate way to proceed with the experiment. Testing functions `fd_obs` and `pfd_obs` below implement continued interaction, simulating the long term effect of the *feedback* and *partial feedback* combinators. The code for the former is straightforward; for the latter, however, one should be aware that an input stream is supplied and, whenever a value of type `Z` is produced, this is appended to the input stream and fed back.

```
def fd_obs: K * cp(K,K)  -> stream(K)
    = (k, p)  =>
      (| (k, p) => head: k
       |           tail: (p1 a(k, p), p0 a(k, p))
       |) (k, p).


def pfd_obs: stream(I+Z) * cp(I+Z,O+Z)  -> stream(O+Z)
    = (s, p)  =>
      (| (s, p) =>
           head: p1 a(head s, p)
       |   tail:  { b0 o => (tail s, p0 a(head s, p))
                  | b1 z => (scons(b1 z, tail s), p0 a(head s, p))
                  } p1 a(head s, p)
       |) (s, p).
```

where the appending function for streams is given as a *record* (§E.5):

```
def scons: X * stream X  -> stream X
    = (x, s)  => (head: x, tail: s).
```

The following printout shows the testing of `conv` with the stream

```
def intnats: int+nat  -> stream(int+nat)
    = x => (| b0 i => head: b0 i
             |         tail: b0 add_int(5,i)
             | b1 n => head: b1 n
             |         tail: b1 succ n
             |) x.
```

forcing the feedback of any output of type `nat`:

```
>> pfd_obs(intnats b0 1, conv 2).
(head: ..., tail: ...)

Right display mode:
(q - quit, return - more) >>
(head: b1(succ(succ(succ(zero)))), tail: (head: ..., tail: ...))

Right display mode:
(q - quit, return - more) >>
(head: b1(succ(succ(succ(zero)))), tail: (head: b0(7), tail: ...))
...
```

87. Generic Components. Let us now generalise $\mathsf{B} = \mathsf{Id}$ to an arbitrary strong monad. In Charity, this generic type of B-behaviours is written as

```
data U -> cp(I,O) =  a : U -> I => B (U * O).
```

where B implements B. The associated functions η, μ, strengths τ_r and τ_l, and the distributive law δ_l, are all explicitly introduced. The price of genericity is, of course, the explicit use of such morphisms in the implementation of the combinators. But, proceeding this way, their implementations become almost transliterations of the formal definitions. Consider, for example, the implementation of *sequential composition*, *choice* and *parallel*:

```
def pipe: cp(I, K) * cp(K, O) -> cp(I, O)
    = pp =>
      (| (p, q) =>
          a: i => mu B{B{iassoc}} B{taul} B{fp{fid,a}}
                  B{fp{fid,swap}} B{assoc} B{xr} taur (a(i,p), q)
       |) pp.

def choice: cp(I, O) * cp(J, R) -> cp(I+J, O+R)
    = pp =>
      (| (p, q) =>
          a: b0 i => B{fp{fid, b0}} B{xr} taur (a(i,p), q)
           | b1 j => B{fp{fid, b1}} B{iassoc} taul (p, a(j,q))
       |) pp.

def par: cp(I, O) * cp(J, R) -> cp(I*J, O*R)
    = pp => (| (p, q) =>
                a: (i,j) => B{m} deltal (a(i,p), a(j,q))
             |) pp.
```

Function *lifting* involves a final application of η and, of course, the unit for **;** arises as a particular lifting — that of the identity. On the other hand, the *wrapping* combinator must apply B to the output wrapping function. Thus,

```
def lift{f:A -> X}: 1 -> cp(A,X)
    = () => (| _ => a: i => eta ((), f i ) |) ().

def copy: 1 -> cp(K,K)
    = () => (| () => a: k => eta ((), k) |) ().

def wrap{f:I' -> I, g:O -> O'}: cp(I, O) -> cp(I',O')
    = pp => (| p => a: i' => B{fp{fid,g}} a(f i', p) |) pp.
```

Finally, the two *feedback* combinators:

```
def feed: cp(K, K) -> cp(K,K)
    = pp => (| p => a: k => mu B{a} B{swap} a(k,p) |) pp.

def pfeed: cp(I+K, O+K) -> cp(I+K, O+K)
    = pp => (| p =>
               a: x =>  mu B{codiag} B{fs{eta,a}} B{fs{fid,swap}}
                        B{fs{fp{fid, b0},fp{fid, b1}}}
                        B{distr} a(x,p)
             |) pp.
```

The only difference between the CHARITY functions above and the formal definitions is the application of the commutativity morphism — swap — to the result of the computation of observer a. In fact, the output types of a and a are swapped: formally, a is the uncurry of a function from U to $\cdots^I$ and, consequently, is typed as $U \times I \longrightarrow \cdots$, for any carrier. However, from the type declaration, the CHARITY system infers a : $I \times$ cp(I,O) $\longrightarrow \cdots$, where cp(I,O) is the carrier $\nu_{I,O}$ of the final coalgebra. Thus, composition with swap is required to feed the result of a back to a.

88. HOUSEKEEPING MORPHISMS. A parenthesis is in order to introduce the implementations of the common 'housekeeping' morphisms used in combinator definitions. *Associativity*, *commutativity*, *distributivity* and *exchange morphisms* are written as

```
def assoc: (X * Y) * Z -> X * (Y * Z)
    = ((x,y),z) => (x,(y,z)).

def iassoc: X * (Y * Z) -> (X * Y) * Z
    = (x,(y,z)) => ((x,y),z).

def swap: X * Y -> Y * X
    = (x,y) => (y,x).

def distr: X * (Y + Z) -> (X * Y) + (X * Z)
    = (x, w) => {b0 y => b0 (x,y)
                |b1 z => b1 (x,z)
                }w .

def xr: (X * Y) * Z -> (X * Z) * Y
    = ((x,y),z) => ((x,z),y).

def m: (X * Y) * (Z * W) -> (X * Z) * (Y * W)
    = ((x,y), (z,w)) => ((x,z), (y,w)).
```

Next consider some common functional combinators in Charity: *function* product and *sum*, *either* and *split* and *identity*. Note, in particular, how function arguments are passed to Charity programs (§E.4).

```
def fp{f:I -> O, g:J -> R}: I * J -> O * R
    = (i,j) => (f i, g j).

def fs{f:I -> O, g:J -> R}: I + J -> O + R
    = b0 i => b0 f i
    | b1 j => b1 g j.

def feither{f:I -> O, g:J -> O}: I + J -> O
    = b0 i =>  f i
    | b1 j =>  g j.

def fsplit{f:I -> O, g:I -> R}: I -> O * R
    = i =>  (f i, g i).

def fid: I -> I
    = i => i.
```

89. Partiality and No Determinism. Instantiating B with different monads gives rise to different classes of components. For example, components whose behaviour is deterministic but partial are obtained with $\mathsf{B}\ X = X + \mathbf{1}$. The Charity implementation of the associated morphisms (unit, multiplication and strengths) is found in appendix A (§13). That is all we have to provide the generic combinators module above to implement this particular class of behaviours.

Nondeterministic behaviours are obtained by instantiating the declaration with the finite powerset monad:

```
data B X = set X.
```

based on a suitable implementation of sets. As CHARITY does not allow a coinductive type to be parametrized by another coinductive type, the definition of sets by characteristic functions discussed in §E.15 cannot be used. We resort instead to a more conventional implementation based on sequences. Monad unit and multiplication are given by

```
def eta: X -> B X
    = a => [a].

def mu: B B X -> B X
    = s => reduce{union, emptyset} s.
```

whereas the generic codification of both strengths and distributive laws used above remains valid. The next paragraph presents an example to illustrate the use of the combinators in this more general setting. We shall revisit the testing strategies, in the general case, in §91.

90. EXAMPLE. Consider the following component

$$\mathsf{circle} : \mathbb{Z} \longrightarrow \mathbb{Z} \quad = \quad \langle 0 \in \mathbb{Z}, \overline{a}_{\mathsf{Circle}} \rangle$$

whose internal state u is a positive integer. Whenever stimulated with another integer n it splits into two configurations corresponding to (and delivering) the integers located at the distance u of n. The value of u is decremented in both cases until it eventually reaches 0. Then the component 'dies' in the sense that it reduces itself to the empty set. Formally,

$$a_{\mathsf{circle}}\ \langle u, n \rangle \ = ((u = 0)\ \rightarrow\ \emptyset,\ \{\langle u-1, n-u \rangle, \langle u-1, n+u \rangle\})$$

The behaviour of circle is computed as the correspondent anamorphism, leading to the following CHARITY implementation:

```
def circle: int -> cp(int,int)
    = u =>
      (| u =>
         a: n =>
          {0 => emptyset
          |_ => [(dec u, sub_int(n,u)), (dec u, add_int(n,u))]
          } u
       |) u.
```

The printout below shows the outcome of some experiments with `circle` in CHARITY. Note, in particular the difference between *sequential composition* — in which the distances to the intermediate inputs are computed in a new component (and a 'fresh' state value) — and *feedback*, in which case such computation is relative to the updated state value.

```
>> a(5, circle 0).
[] : set(cp(int, int) * int)

>> a(5, circle 2).
[((a: <function>), 3), ((a: <function>), 7)]
 : set(cp(int, int) * int)

>> a(5, pipe (circle 2, circle 2)).
[((a: <function>), 1), ((a: <function>), 5),
 ((a: <function>), 5), ((a: <function>), 9)]
 : set(cp(int, int) * int)

>> a(5, feed circle 2).
[((a: <function>), 2), ((a: <function>), 4),
  ((a: <function>), 6), ((a: <function>), 8)]
 : set(cp(int, int) * int)

>> a( b1 4, choice (circle 5, circle 7)).
[((a: <function>), b1(-3)), ((a: <function>), b1(11))]
 : set(cp((int + int), (int + int)) * (int + int))
```

The set of outputs produced as a reaction to a particular stimulus can be easily accessed by (the functorial image of) the relevant projections, and, similarly, for the 'continuation' behaviours. For example,

```
>> B{p1} a((2,4), par (circle 5, circle 7)).
[(-3, -3), (-3, 11), (7, -3), (7, 11)] : set(int * int)

>> B{p0} a((2,4), par (circle 5, circle 7)).
[(a: <function>), (a: <function>), (a: <function>), (a: <function>)]
  : set(cp(int * int, int * int))
```

91. Testing Strategies Revisited. Tracing the behaviour of a nondeterministic component involves a suitable selection of ‘continuation points’. If in the deterministic case a simple projection is enough, here a ‘pick one from a set’ operation is required. For example, in

```
>> a((8,-4), pick B{p0} a((2,4), par (circle 5, circle 7))).
[((a: <function>), (4, -10)), ((a: <function>), (4, 2)),
 ((a: <function>), (12, -10)), ((a: <function>), (12, 2))]
 : set(cp((int * int), (int * int)) * (int * int))
```

`pick` selects, based on a random number generator, the ‘second’ continuation produced by the parallel composition of `circle 5` and `circle 7` on reaction to $\langle 2, 4\rangle$. A similar ‘access’ function is required for partial behaviours. In this case, it has to find out if a continuation has been produced. The following function does the job:

```
def pick_if_available: I * SF cp(I,J) -> SF (cp(I,J) * J)
    = (i, ff) => ff | (i, ss p) => a(i,p).
```

In general, ‘access’ functions are specific to the underlying behaviour monad. Also dependent on the behaviour model are the ‘iteration’ testing functions: in particular, each sort of behaviour requires an appropriate *observation* structure. As discussed above, *streams* provide such a structure for the deterministic case. For partial behaviours, observations can be collected into a finite or infinite sequence, known as a `colist` in Charity (see §E.15). A coinductive definition of the correspondent testing function follows:

```
def obs: stream I * cp(I,O) -> colist O
    = (s,p)  =>
      (| (s,p) => delist: { ff => ff
                          | ss(q,o) => ss (o, (tail s, q))
                          }  a(head s, p)
       |) (s,p).
```

The observation structure required by the nondeterministic case is a possibly infinite generalized tree. We define this in Charity as

```
data T -> cotree X = cot: T -> set(X * T).
```

The testing function arises as an anamorphism in the following commuting diagram:

$$\begin{tikzcd}
\texttt{cotree O} \arrow[r, "\omega_{\mathcal{P}(\texttt{O}\times\mathsf{Id})}"] & \mathcal{P}(\texttt{O} \times \texttt{cotree O}) \\
\texttt{stream I} \times \texttt{cp(I,O)} \arrow[u, "\texttt{obs}=[\!(\texttt{obs\_gen})\!]"] \arrow[r, "\texttt{obs\_gen}"] & \mathcal{P}(\texttt{O} \times (\texttt{stream I} \times \texttt{cp(I,O)})) \arrow[u, "\mathsf{id}\times\texttt{obs}"']
\end{tikzcd}$$

where obs_gen is the 'gene' coalgebra:

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Ba} \cdot \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{xr} \cdot (\langle \mathtt{head}, \mathtt{tail} \rangle \times \mathsf{id})$$

leading to the following CHARITY implementation:

```
def obs: stream I * cp(I,O) -> cotree O
    = (s,p)  =>
      (| (s,p) => cot: B{fp{fid,swap}} B{assoc}
                       B{fp{swap,fid}} taur
                       (a(head s, p), tail s)
       |) (s, p).
```

For example, one may observe the reaction of circle 5 to the stream of even numbers computing

```
>> obs(even, circle 5).
(cot: ...)

Right display mode:
(q - quit, return - more) >>
(cot: [(-3, (cot: ...)), (7, (cot: ...))])

Right display mode:
(q - quit, return - more) >>
(cot: [(-3, (cot: [(0, (cot: ...)), (8, (cot: ...))])),
 (7, (cot: [(0, (cot: ...)), (8, (cot: ...))]))])

Right display mode:
(q - quit, return - more) >>
(cot: [(-3, (cot: [(0, (cot: [(5, (cot: ...)), (11, (cot: ...))])),
 (8, (cot: [(5, (cot: ...)), (11, (cot: ...))]))])),
 (7, (cot: [(0, (cot: [(5, (cot: ...)), (11, (cot: ...))])),
 (8, (cot: [(5, (cot: ...)), (11, (cot: ...))]))]))])

...
```

Notice this is, of course, just the behaviour up to bisimulation, as the implementation of sets used here was based on sequences.

CHAPTER 6

Components as Objects

Summary

This chapter addresses an alternative model for components, closer to the object-orientation paradigm, based on a class of functors whose shape does not entail any input output dependence. The component algebra introduced in the previous chapter is revisited in this new setting. The emphasis is put, however, in a category with components as objects, rather than on the bicategorical structure. The chapter includes a discussion of how component internal activity can be dealt within this model. A mild generalisation of a component morphism using monadic wiring functions is briefly discussed.

1. An Alternative Model

1. This chapter investigates an alternative model for components in which there is no immediate dependence between input stimuli and the outputs produced. The relevant functor is

$$\mathsf{T}^{\mathsf{B}} \;=\; O \times \mathsf{B}^{I} \tag{6.1}$$

where I and O are interface types and B is a strong monad, as before. A coalgebra for this functor can be written as the split of two functions

$$\langle o_p, \overline{a_p} \rangle : U_p \longrightarrow O \times (\mathsf{B}\; U_p)^{I}$$

where $o_p : U_p \longrightarrow O$ is a state observer (usually called the *attribute* in the object-oriented programming paradigm) and $a_p : U_p \times I \longrightarrow \mathsf{B}U_p$ is a state update function (usually referred to as the *method* or *action*). Note that often O instantiates to a Cartesian product $\prod_{x \in X} O_x$ of different, but simultaneously available, observers, whereas I takes the form of a sum $\sum_{y \in Y} I_y$ of (state update, non interfering) operations. If Y

is regarded as a set of action names, I_y denotes the type of the argument of operation y.

The comorphism condition for this functor is derived from the general case (§3.20). Thus, $h : p \longrightarrow q$ is a comorphism from p to q iff

$$\begin{array}{ll} & (O \times \mathsf{B}h^I) \cdot p \;=\; q \cdot h \\ \equiv & \qquad \{ \text{ definition } \} \\ & o_p = o_q \cdot h \;\wedge\; \mathsf{B}h^I \cdot \overline{a_p} = \overline{a_q} \cdot h \\ \equiv & \qquad \{ \text{ exponential fusion and absorption } \} \\ & o_p = o_q \cdot h \;\wedge\; \overline{\mathsf{B}h \cdot a_p} = \overline{a_q \cdot (h \times \mathsf{id})} \\ \equiv & \qquad \{ \text{ exponential universal } \} \\ & o_p = o_q \cdot h \;\wedge\; \mathsf{B}h \cdot a_p = a_q \cdot (h \times \mathsf{id}) \end{array}$$

Components modeled in this way will be called, in the sequel, *object components*, after their similarity with (a simple) notion of an object in object-oriented programming. For instance, [Jac96b] uses this functor, with $\mathsf{B} = \mathsf{Id}$, in a coalgebraic model for the object paradigm.

Behaviours of *object* components are animated in CHARITY along the same lines discussed in the previous chapter. In particular, they are declared as inhabitants of the following coinductive type

```
data S -> obj(I,O) = ob : U -> O
                   | ac : U -> I => B U.
```

parametric on the input (`I`) and output (`O`) types, where `B` denotes a monad specification. The same applies, of course, to the component algebra which will be revisited in the following sections. However, for space economy, we refrain from presenting prototyping details.

2. ELEMENTARY EXAMPLES. A *queue* is a simple example of an object component with two observers (top and $\mathsf{isempty?}$) and two actions (enq and deq). As deq is partial, the use of the maybe monad seems adequate to force deadlock whenever an illegal deq is performed. Let E be a set and take sequences of E as the state space. Then, define

$$\mathsf{Queue} : E + \mathbf{1} \longrightarrow (E + \mathbf{1}) \times \mathbf{2}$$

as

$$\mathsf{Queue} = \langle <> \in E^*,\ \langle o_{\mathsf{Queue}}, \overline{a}_{\mathsf{Queue}} \rangle : E^* \longrightarrow ((E + \mathbf{1}) \times \mathbf{2}) \times (E^* + \mathbf{1})^{E+\mathbf{1}} \rangle$$

and the operations in the usual way:

$$\begin{aligned} o_{\mathsf{Queue}} &= \langle \mathsf{top}, \mathsf{isempty?} \rangle \\ \text{where}\quad \mathsf{top}\, s &= (s = <> \to \iota_2 *,\ \iota_1(\mathsf{last}\, s)\ ) \\ \mathsf{isempty?}\ s &= s = <> \\ a_{\mathsf{Queue}} &= [\mathsf{enq}, \mathsf{deq}] \cdot \mathsf{dl} \\ \text{where}\quad \mathsf{enq}\ (s,e) &= \iota_1\ (s \frown < e >) \\ \mathsf{deq}\ (s,*) &= (s = <> \to \iota_2 *,\ \iota_1\ (\mathsf{blast}\, s)) \end{aligned}$$

where, given a sequence s, $\mathsf{blast}\, s$ returns s without its last element. Notice that deq has a dummy parameter (of type $\mathbf{1}$) made explicit in the component interface to represent a *trigger* for this action.

Another example of a simple component is the specification of a finite, nondeterministic automaton corresponding to a regular language, for example $E = \alpha\, \beta\, E + \beta + \beta\, E$. In this case the powerset monad is the appropriate choice for B. The attribute is taken as boolean-valued to discriminate between final and intermediate states (*i.e.*, 'parsing' stages). Let, thus, $T = \{\alpha, \beta, ...\}$ be a set of terminals and Exp the set of regular expressions over T. Then define

$$\mathsf{A} : T \longrightarrow \mathbf{2}$$

given by

$$\mathsf{A}\ =\ \langle E \in Exp,\ \langle o_{\mathsf{A}}, \overline{a_{\mathsf{A}}} \rangle : Exp \longrightarrow \mathbf{2} \times (\mathcal{P} Exp)^T \rangle$$

where $o_{\mathsf{A}} = (=_\epsilon)$ and a_{A} is given by the following clauses,

$$\begin{aligned} a_{\mathsf{A}}(E, \alpha) &= \{\beta\, E\} \\ a_{\mathsf{A}}(E, \beta) &= \{E, \epsilon\} \\ a_{\mathsf{A}}(\beta\, E, \beta) &= \{E\} \\ a_{\mathsf{A}}(e, t) &= \emptyset \qquad \text{for all other } e \in Exp \text{ and } t \in T \end{aligned}$$

3. COMPOSITION. The examples above illustrate *object* components, parametric on input type I and output O and defined as seeded coalgebras for functor T^{B} (6.1). The definition of *sequential* composition given in §5.2 can be adapted to this new setting, although, as shown below, one of its properties requires careful consideration. Let $p : I \longrightarrow K$ and $q : K \longrightarrow O$ be two components. Their composition is formed by placing them side by side and connecting the output attribute of p to the input of q action. Formally, $p\ ;\ q : I \longrightarrow O$ is given by

$$p\ ;\ q\ =\ \langle \langle u_p, u_q \rangle \in U_p \times U_q, \langle o_{p;q}, \overline{a}_{p;q} \rangle \rangle$$

$$
\begin{aligned}
o_{p;q} &= U_p \times U_q \xrightarrow{\pi_2} U_q \xrightarrow{o_q} O \\
a_{p;q} &= U_p \times U_q \times I \xrightarrow{\mathsf{xr}} U_p \times I \times U_q \xrightarrow{a_p \times \mathsf{id}} \mathsf{B}U_p \times U_q \xrightarrow{\tau_r} \mathsf{B}(U_p \times U_q) \\
&\xrightarrow{\mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle} \mathsf{B}(U_p \times U_q \times K) \xrightarrow{\mathsf{Ba}} \mathsf{B}(U_p \times (U_q \times K)) \\
&\xrightarrow{\mathsf{B}(\mathsf{id} \times a_q)} \mathsf{B}(U_p \times \mathsf{B}U_q) \xrightarrow{\mathsf{B}\tau_l} \mathsf{BB}(U_p \times U_q) \xrightarrow{\mu} \mathsf{B}(U_p \times U_q)
\end{aligned}
$$

Clearly, sequential composition is associative, *i.e.*, for p, q and r suitably typed

$$
p\,;q\,;r \;\sim\; p\,;(q\,;r) \tag{6.2}
$$

Proof. The bisimulation is witnessed by $\mathsf{a} : U_p \times U_q \times U_r \longrightarrow U_p \times (U_q \times U_r)$. To prove that a is a comorphism, consider first the observers part:

$$
\begin{aligned}
& o_{(p;q);r} \\
= & \quad \{ \ ; \text{definition} \ \} \\
& o_r \cdot \pi_2 \\
= & \quad \{ \ \text{routine: } \pi_2 \cdot \pi_2 \cdot \mathsf{a} = \pi_2 \ \} \\
& o_r \cdot \pi_2 \cdot \pi_2 \cdot \mathsf{a} \\
= & \quad \{ \ ; \text{definition} \ \} \\
& o_{q;r} \cdot \pi_2 \cdot \mathsf{a} \\
= & \quad \{ \ ; \text{definition} \ \} \\
& o_{p;(q;r)} \cdot \mathsf{a}
\end{aligned}
$$

Concerning the action part, the proof is similar to that of the corresponding property for *functional* components in §5.4. The only difference here is that, instead of using the intermediate result produced by p, this is obtained by computing o_p over the state space of p. However, once o_p is computed, the state of p has already been updated by a_p.

□

To investigate the existence of units for **;** requires a prior introduction of a mechanism for representing functions as components (*cf.*, the procedure followed in the previous chapter). This is done in next paragraph.

4. Representation of Functions. A function $f : A \longrightarrow B$ can be represented as an object component via the following '*mirror* mechanism':

$$
\ulcorner f \urcorner \;=\; \langle b \in B, \langle \mathsf{id}_B, \overline{\eta_B \cdot f \cdot \pi_2} \rangle \rangle
$$

i.e., by a coalgebra $B \longrightarrow B \times (\mathsf{B}B)^A$. Note that, in general, the representation of f is not unique. Even worse, the possible representations are not bisimilar. We shall come back to this soon. For the moment, consider

$$\mathsf{copy}_K \;=\; \ulcorner \mathsf{id}_K \urcorner$$

Is this a unit for sequential composition?

5. UNIT. Let $p : I \longrightarrow O$ be a component. We want to discuss whether equations

$$\mathsf{copy}_I \,; p \;\sim\; p \tag{6.3}$$

$$p \,; \mathsf{copy}_O \;\sim\; p \tag{6.4}$$

hold. The obvious choice of comorphisms to witness bisimulations in equations (6.3) and (6.4) is, respectively, $\pi_2 : I \times U_p \longrightarrow U_p$ and $\pi_1 : U_p \times O \longrightarrow U_p$. That, in both cases, such comorphisms commute with the action part of the respective components is proved in [Appendix D, page 396]. For the observers, however, one has:

$$o_{\mathsf{copy}_I;p} \;=\; o_p \cdot \pi_2$$

but

$$o_{p;\mathsf{copy}_O} \;=\; o_{\mathsf{copy}_O} \cdot \pi_2$$

which, in general, differs from $o_p \cdot \pi_1$. Anyway, should both p ; copy_O and p have been observed after action takes place, the result would be the same, because the 'expected' value would be already stored in the state of copy_O. This leads to the following definition of a *next comorphism* which, in a sense, retrieves the 'functional dependence' scheme.

6. DEFINITION. A *next comorphism* from $p : I \longrightarrow O$ to $q : I \longrightarrow O$ is a function $h : U_p \longrightarrow U_q$ such that

$$\mathsf{B}o_p \cdot a_p \;= \mathsf{B}o_q \cdot a_q \cdot (h \times \mathsf{id}) \tag{6.5}$$

and

$$\mathsf{B}h \cdot a_p \;=\; a_q \cdot (h \times \mathsf{id}) \tag{6.6}$$

Two components p and q are *next bisimilar*, written as $p \overset{\bullet}{\sim} q$ iff there is a seed preserving next comorphism $h : p \longrightarrow q$ relating them.

7. LEMMA. The composition of next comorphisms is a next comorphism as well as identity is its unit. Moreover, every comorphism is also a next comorphism.

Proof. The identity case is trivial. For composition let $h : p \longrightarrow q$ and $k : q \longrightarrow r$ be next comorphisms. Then,

$$\begin{aligned}
& a_r \cdot ((k \cdot h) \times \mathsf{id}) \\
= \quad & \quad \{ \ \times \text{ functor and } k \text{ is a next comorphism } \} \\
& \mathsf{B}k \cdot a_q \cdot (h \times \mathsf{id}) \\
= \quad & \quad \{ \ \mathsf{B} \text{ functor and } h \text{ is a next comorphism } \} \\
& \mathsf{B}(k \cdot h) \cdot a_p
\end{aligned}$$

and

$$\begin{aligned}
& \mathsf{B}o_p \cdot a_p \\
= \quad & \quad \{ \ h \text{ is a next comorphism } \} \\
& \mathsf{B}o_q \cdot a_q \cdot (h \times \mathsf{id}) \\
= \quad & \quad \{ \ k \text{ is a next comorphism } \} \\
& \mathsf{B}o_r \cdot a_r \cdot (k \times \mathsf{id}) \cdot (h \times \mathsf{id}) \\
= \quad & \quad \{ \ \times \text{ functor } \} \\
& \mathsf{B}o_r \cdot a_r \cdot ((k \cdot h) \times \mathsf{id})
\end{aligned}$$

Every comorphism is a next comorphism: if $h : p \longrightarrow q$ is a comorphism it already meets (6.6). It furthermore satisfies (6.5) because

$$\begin{aligned}
& \mathsf{B}o_p \cdot a_p \\
= \quad & \quad \{ \ h \text{ is a comorphism (attribute condition) and } \mathsf{B} \text{ functor } \} \\
& \mathsf{B}o_q \cdot \mathsf{B}h \cdot a_p \\
= \quad & \quad \{ \ h \text{ is a comorphism (action condition) } \} \\
& \mathsf{B}o_q \cdot a_q \cdot (h \times \mathsf{id})
\end{aligned}$$

□

8. REMARK. What we have termed above *next bisimilarity* and *next comorphism* are just the usual notions of bisimilarity and comorphism for functor $\mathsf{R} = \mathsf{B}\mathcal{O}^{I} \times \mathsf{B}^{I}$. It is immediate to check that

$$p \overset{\bullet}{\sim} q \quad \text{iff} \quad \overset{\bullet}{p} \sim \overset{\bullet}{q}$$

where $\overset{\bullet}{p} = \langle u_p \in U_p, \langle \overline{\mathsf{B}o_p \cdot a_p}, \overline{a_p} \rangle \rangle$ and $\sim$ on the righthandside means R-bisimilarity. We may now check that equation (6.4) holds if stated in terms of next bisimilarity:

9. LEMMA. Let $p : I \longrightarrow O$ be a component. Then

$$p \,;\, \mathsf{copy}_O \overset{\bullet}{\sim} p \tag{6.7}$$

Proof. The action part for $\sim$ has already been dealt with in §5. By the second part of lemma §7, this remains valid for $\overset{\bullet}{\sim}$. Therefore, it remains to check the (critical) observers part:

$$\begin{aligned}
& \mathsf{B}o_{p;\mathsf{copy}_O} \cdot a_{p;\mathsf{copy}_O} \\
= \quad & \{ \ ; \text{ definition} \} \\
& \mathsf{B}o_{(\mathsf{copy}_O} \cdot \pi_2) \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{\mathsf{copy}_O}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mathsf{copy}_O \text{ definition} \} \\
& \mathsf{B}\pi_2 \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \eta) \cdot \mathsf{B}(\mathsf{id} \times \pi_2) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \text{law (C.20)} \} \\
& \mathsf{B}\pi_2 \cdot \mu \cdot \eta \cdot \mathsf{B}(\mathsf{id} \times \pi_2) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \text{law (C.14)} \} \\
& \mathsf{B}\pi_2 \cdot \mathsf{B}(\mathsf{id} \times \pi_2) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \text{routine: } \pi_2 \cdot (\mathsf{id} \times \pi_2) \cdot \mathsf{a} = \pi_2 \ \} \\
& \mathsf{B}\pi_2 \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \times \text{ cancellation} \} \\
& \mathsf{B}(o_p \cdot \pi_1) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \text{law (C.12)} \} \\
& \mathsf{B}o_p \cdot \pi_1 \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \times \text{ cancellation} \} \\
& \mathsf{B}o_p \cdot a_p \cdot \pi_1 \cdot \mathsf{xr} \\
= \quad & \{ \ \text{routine: } \pi_1 \cdot \mathsf{xr} = \pi_1 \times \mathsf{id} \} \\
& \mathsf{B}o_p \cdot a_p \cdot (\pi_1 \times \mathsf{id})
\end{aligned}$$

□

10. SEEDS AS PREDICATES. Let p_1 and p_2 be two liftings of function $f : A \longrightarrow B$ over different seeds b_1 and b_2 in B. Clearly, $p_1 \overset{\bullet}{\sim} p_2$, witnessed by id_B, but id_B is not seed preserving — actually there is no next comorphism relating p_1 and p_2 and preserving seeds. This observation motivates a broader definition of a component, replacing the seed value $u_p \in U_p$ by a predicate over U_p intended to characterise all possible initial states for p. Therefore, we define an *object* component as

$$p \;=\; \langle \gamma_p : U_p \longrightarrow \mathbf{2}, \langle o_p : U_p \longrightarrow O, \overline{a_p} : U_p \longrightarrow \mathsf{B}U_p^I \rangle\rangle \qquad (6.8)$$

where γ_p is named the *seed predicate*. Both comorphisms and next comorphisms are then required to satisfy the following seed predicate preserving condition:

$$\gamma_p \;=\; \gamma_q \cdot h \qquad (6.9)$$

and the definitions of sequential composition and function lifting modified to include, respectively,

$$\gamma_{p;q} \;=\; \wedge \cdot (\gamma_p \times \gamma_q)$$

and

$$\gamma_{\ulcorner f \urcorner} \;=\; \underline{\mathsf{true}} \cdot !$$

The fact that U_p is not empty is required in this model. In fact, the empty state space would turn the homsets in the corresponding category of behaviours into singletons: by initiality there would be a comorphism from the component with empty state space to every other one. As a result all components with the same interface would be bisimilar.

11. BEHAVIOUR. Given a component $p : I \longrightarrow O$ and a valid element of its state space, *i.e.*, $u \in U_p$ such that $\gamma_p\ u$, the *behaviour* of p at u is computed by coinductive extension, *i.e.*,

$$[\![\langle p, u \rangle]\!] \;=\; [\![\langle o_p, \overline{a}_p \rangle]\!] u$$

12. LEMMA. Let B be a strong monad. Then a bicategory Cp_B is formed taking sets as objects, object components, defined by (6.8), as arrows and next comorphisms as 2-cells.

Proof. The proof follows the argument used to verify the corresponding result for *functional* components in §5.4. First note that each $\mathsf{Cp}_\mathsf{B}(I, O)$ is a full subcategory of the category of R coalgebras over Set, for $\mathsf{R}\ X = \mathsf{B}O^I \times \mathsf{B}X^I$. More precisely, its restriction to coalgebras of the form $\overset{\bullet}{p}$, for p a coalgebra for functor (6.1). Then, for objects I, O and K, both sequential composition and its unit can be made into a family of functors $;_{I,K,O}$ and copy_K by defining its action on 2-cells as, respectively, function product (*i.e.*, $h \;;_{I,K,O}\; k = h \times k$) and identity on K (*i.e.*, $\mathsf{copy}_K\ \mathsf{id}_* = \mathsf{id}_K$). Checking the functoriality axioms is immediate. However, the step involved in checking that $h \times k$ is indeed a next comorphism is non trivial. This is proved in [Appendix D, page 398]. Finally, the existence of natural isomorphisms expressing

; associativity, left and right units, has already been established in §3, §5 and §9. Being also Set natural isomorphisms, coherence conditions hold trivially.

□

2. Component Algebra Revisited

13. STEPPING DOWN. This section revisits the combinators introduced in the previous chapter in the new setting of *object* components. For this purpose, one could resort to the bicategorical structure already defined and introduce such operators as lax functors, as before. To pave the way to a possible generalisation of component morphisms, proposed later in this chapter, we shall frame our discussion in a category $\mathsf{St_B}$ having components as objects and seed predicate preserving comorphisms as arrows. Notice that a category with components as objects can always be obtained from $\mathsf{Cp_B}$ by application of the general 'stepping down' procedure in bicategories (§B.11): arrows become objects and 2-cells the arrows of the new category.

Our starting point is an expected result on function lifting, which corresponds to lemma §5.11.

14. LEMMA. Let $g : I \longrightarrow K$ and $f : K \longrightarrow O$ be functions. Then,

$$\ulcorner f \cdot g \urcorner \sim \ulcorner g \urcorner \,;\, \ulcorner f \urcorner \tag{6.10}$$

$$\ulcorner \mathsf{id}_I \urcorner \sim \mathsf{copy}_I \tag{6.11}$$

Proof. Equation (6.11) is an immediate consequence of the definition and can therefore be written as an equality. Equation (6.10) will be proved by checking that $\pi_2 : K \times O \longrightarrow O$ is a comorphism from $\ulcorner g \urcorner \,;\, \ulcorner f \urcorner$ to $\ulcorner f \cdot g \urcorner$. For the observers' part note that

$$o_{\ulcorner g \urcorner ; \ulcorner f \urcorner} \;=\; \pi_2 \cdot \mathsf{id}_O \;=\; o_{\ulcorner f \cdot g \urcorner}$$

holds. On the other hand,

$$
\begin{array}{cl}
 & \mathsf{B}\pi_2 \cdot a_{\ulcorner g \urcorner ; \ulcorner f \urcorner} \\
= & \qquad \{ \text{ ; and function lifting definitions } \} \\
 & \mathsf{B}\pi_2 \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times (\eta \cdot f \cdot \pi_2)) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot g \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ law (C.20) } \} \\
 & \mathsf{B}\pi_2 \cdot \mu \cdot \eta \cdot \mathsf{B}(\mathsf{id} \times (f \cdot \pi_2)) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot g \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ law (C.14) } \} \\
 & \mathsf{B}\pi_2 \cdot \mathsf{B}(\mathsf{id} \times (f \cdot \pi_2)) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot g \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr}
\end{array}
$$

$$
\begin{aligned}
= \quad & \{ \text{ routine: } (\mathsf{id} \times \pi_2) \cdot \mathsf{a} = \pi_1 \times \mathsf{id} \, \} \\
& \mathsf{B}\pi_2 \cdot \mathsf{B}(\mathsf{id} \times f) \cdot \mathsf{B}(\pi_1 \times \mathsf{id}) \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot g \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \times \text{ absorption } \} \\
& \mathsf{B}\pi_2 \cdot \mathsf{B}\langle \pi_1, f \cdot \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot g \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \times \text{ cancellation } \} \\
& \mathsf{B}f \cdot \mathsf{B}\pi_1 \cdot \tau_r \cdot ((\eta \cdot g \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \text{ law (C.18) } \} \\
& \mathsf{B}f \cdot \mathsf{B}\pi_1 \cdot \eta \cdot (g \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \eta \text{ natural (C.17) } \} \\
& \eta \cdot f \cdot \pi_1 \cdot (g \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \; \times \text{ cancellation } \} \\
& \eta \cdot f \cdot g \cdot \pi_1 \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \text{ routine: } \pi_1 \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} = \pi_2 \cdot (\pi_2 \times \mathsf{id}) \, \} \\
& \eta \cdot f \cdot g \cdot \pi_2 \cdot (\pi_2 \times \mathsf{id}) \\
= \quad & \{ \text{ function lifting definition } \} \\
& \ulcorner f \cdot g \urcorner \cdot (\pi_2 \times \mathsf{id})
\end{aligned}
$$

□

15. The next paragraphs introduce *wrapping*, *parallel*, *choice* and *concurrent* aggregation. Note the similarity with the corresponding definitions in chapter 5, although some combinators become differently typed on account of the availability of the observers. Therefore, whatever combinator is used to aggregate two components, the observers of both arguments are simultaneously available and the output of the aggregated component becomes a product. We shall be rather sketchy about the properties of those combinators — in most cases they are similar to the ones of the corresponding *functional* components' algebra. Their proofs are also similar but for the observers, which require separate treatment.

16. WRAPPING. The *wrapping* combinator encapsulates component's pre- and post-composition with function lifting. Formally, for $p : I \longrightarrow O$ an object component and functions $f : I' \longrightarrow I, g : O \longrightarrow O'$, define

$$-[f,g] : \mathsf{Cp}(I,O) \longrightarrow \mathsf{Cp}(I',O')$$

mapping $\langle \gamma_p, \langle o_p, \overline{a}_p \rangle\rangle$ into $\langle \gamma_p, \langle o_{p[f,g]}, \overline{a}_{p[f,g]} \rangle\rangle$, where

$$\begin{aligned} o_{p[f,g]} &= U_p \xrightarrow{o_p} O \xrightarrow{g} O' \\ a_{p[f,g]} &= U_p \times I' \xrightarrow{\mathsf{id} \times f} U_p \times I \xrightarrow{a_p} \mathsf{B}U \end{aligned}$$

17. LEMMA. For any component $p : I \longrightarrow O$ and functions $f : I' \longrightarrow I, g : O \longrightarrow O', f' : J \longrightarrow I'$ and $g' : O' \longrightarrow R$,

$$p[f,g][f',g'] \overset{\bullet}{\sim} p[f \cdot f', g' \cdot g] \tag{6.12}$$

$$p[f,g] \overset{\bullet}{\sim} \ulcorner f \urcorner \,;\, p \,;\, \ulcorner g \urcorner \tag{6.13}$$

Proof. The proof follows the arguments used in §5.19 and §5.20. Note, however, the need for introducing next bisimilarity exactly by the same reason why copy fails to be a unit for sequential composition under the usual bisimilarity relation. Concerning equation (6.12) it is easy to check that $p[f, \mathsf{id}] \sim \ulcorner f \urcorner ; p$ holds, but, on the other hand, one only gets $p[\mathsf{id}, g] \overset{\bullet}{\sim} p ; \ulcorner g \urcorner$. □

18. PARALLEL. Synchronous product is the simplest form of component aggregation: given $p : I \longrightarrow O$ and $q : J \longrightarrow R$, $p \boxtimes q$ executes simultaneously both actions and offers the product of both attributes. Formally,

$$p \boxtimes q = \langle \gamma_{p \boxtimes q}, \langle o_{p \boxtimes q}, \overline{a}_{p \boxtimes q} \rangle\rangle$$

where

$$\begin{aligned} \gamma_{p \boxtimes q} &= U_p \times U_q \xrightarrow{\gamma_p \times \gamma_q} \mathbf{2} \times \mathbf{2} \xrightarrow{\wedge} \mathbf{2} \\ o_{p \boxtimes q} &= U_p \times U_q \xrightarrow{o_p \times o_q} O \times R \end{aligned}$$

and

$$\begin{aligned} a_{p \boxtimes q} &= U_p \times U_q \times (I \times J) \xrightarrow{\mathsf{m}} U_p \times I \times (U_q \times J) \\ &\xrightarrow{a_p \times a_q} \mathsf{B}U_p \times \mathsf{B}U_q \xrightarrow{\delta_l} \mathsf{B}(U_p \times U_q) \end{aligned}$$

Its unit is, as in the case of *functional* components,

$$\mathsf{idle} = \ulcorner \mathsf{id}_1 \urcorner$$

19. CHOICE AND CONCURRENT COMPOSITION. Let $p : I \longrightarrow O$ and $q : J \longrightarrow R$ be two object components. With respect to the action part, $p \boxplus q$ behaves either as p or q depending on the input. On the other hand, $p \boxast q$ allows the selection between an independent action of p or q, and their simultaneous execution. As expected, the combinator captures the intuition that, putting both components side by side, leads to an increase of (observable) behaviour: not only the individual observers and actions of both processes are available, but also there is the possibility of activating them concurrently. As discussed above, the attribute of the composed system, in both cases, is simply the product of o_p and o_q. Formally, *choice* is defined as

$$p \boxplus q \;=\; \langle \gamma_{p \boxplus q}, \langle o_{p \boxplus q}, \overline{a}_{p \boxplus q} \rangle \rangle$$

where

$$\begin{aligned}
\gamma_{p \boxplus q} \;&=\; U_p \times U_q \xrightarrow{\gamma_p \times \gamma_q} \mathbf{2} \times \mathbf{2} \xrightarrow{\wedge} \mathbf{2} \\
o_{p \boxplus q} \;&=\; U_p \times U_q \xrightarrow{o_p \times o_q} O \times R
\end{aligned}$$

and

$$\begin{aligned}
a_{p \boxplus q} \;=\; & U_p \times U_q \times (I + J) \xrightarrow{\mathsf{dr}} U_p \times U_q \times I + U_p \times U_q \times J \\
& \xrightarrow{\mathsf{xr}+\mathsf{a}} U_p \times I \times U_q + U_p \times (U_q \times J) \xrightarrow{a_p \times \mathsf{id} + \mathsf{id} \times a_q} \mathsf{B}U_p \times U_q + U_p \times \mathsf{B}U_q \\
& \xrightarrow{\tau_r + \tau_l} \mathsf{B}(U_p \times U_q) + \mathsf{B}(U_p \times U_q) \xrightarrow{\nabla} \mathsf{B}(U_p \times U_q)
\end{aligned}$$

whereas, *concurrent* composition is given by

$$p \boxast q \;=\; \langle \gamma_{p \boxast q}, \langle o_{p \boxast q}, \overline{a}_{p \boxast q} \rangle \rangle$$

where

$$\begin{aligned}
\gamma_{p \boxast q} \;&=\; U_p \times U_q \xrightarrow{\gamma_p \times \gamma_q} \mathbf{2} \times \mathbf{2} \xrightarrow{\wedge} \mathbf{2} \\
o_{p \boxast q} \;&=\; U_p \times U_q \xrightarrow{o_p \times o_q} O \times R
\end{aligned}$$

and

$$\begin{aligned}
a_{p \boxast q} \;=\; & U_p \times U_q \times (I \boxast J) \xrightarrow{\mathsf{dr}} U_p \times U_q \times (I + J) + U_p \times U_q \times (I \times J) \\
& \xrightarrow{[a_{p \boxplus q}, a_{p \boxtimes q}]} \mathsf{B}(U_p \times U_q)
\end{aligned}$$

Note that both $\boxplus$ and $\boxast$ have the same unit

$$\mathsf{nil} \;=\; \ulcorner !_{\emptyset} \urcorner$$

20. LAWS. Combinators $\boxtimes$, $\boxplus$ and $\boxast$ verify the same laws, formulated as bisimilarity equations, as the corresponding versions for *functional* components. In particular, they commute with sequential composition and, up to isomorphic wiring, form Abelian monoids. The commutativity of the underlying behaviour monad is also assumed for some properties of $\boxtimes$ and $\boxast$.

The proofs are similar to the ones of lemmas §5.24, §5.26, §5.35, §5.37, §5.46 and §5.47 and will be omitted here. In particular, the comorphisms chosen to witness bisimilarity, in each case, are the same. However, as the outputs of $\boxplus$ and $\boxast$ are products, *e.g.*, $O \times R$, instead of sums, *e.g.*, $O + R$ or $O \boxast R$, the required wiring is different in some laws. In particular, the correspondents to equations (5.18), (5.19), (5.20) and (5.21) for $\boxplus$, as well as to (5.49), (5.19), (5.20) and (5.21), for $\boxast$, become, respectively,

$$(p \boxplus q) \boxplus r \sim (p \boxplus (q \boxplus r))[\mathsf{a}_+, \mathsf{a}^\circ] \tag{6.14}$$

$$\mathsf{nil} \boxplus p \sim p[\mathsf{r}_+, \mathsf{r}^\circ] \tag{6.15}$$

$$p \boxplus \mathsf{nil} \sim p[\mathsf{l}_+, \mathsf{l}^\circ] \tag{6.16}$$

$$p \boxplus q \sim (q \boxplus p)[\mathsf{s}_+, \mathsf{s}] \tag{6.17}$$

$$(p \boxast q) \boxast r \sim (p \boxast (q \boxast r))[\mathsf{a}_*, \mathsf{a}^\circ] \tag{6.18}$$

$$p \boxast q \sim (q \boxast p)[\mathsf{s}_*, \mathsf{s}] \tag{6.19}$$

$$\mathsf{nil} \boxast p \sim p[\mathsf{r}_*, \mathsf{r}^\circ] \tag{6.20}$$

$$p \boxast \mathsf{nil} \sim p[\mathsf{l}_*, \mathsf{l}^\circ] \tag{6.21}$$

Again, the unit for $\boxplus$ acts as a zero for $\boxtimes$. However, as nil is now typed as $\emptyset \longrightarrow \mathbf{1}$, the law corresponding to equation (5.36) has to be written as

$$\mathsf{nil} \boxtimes p \sim \mathsf{nil}[\mathsf{zl}, \mathsf{r}^\circ] \tag{6.22}$$

As a final remark on proofs, note that the attribute parts have to be dealt separately and preservation of the seed predicate by the witnessing comorphism has to be ensured. In all cases, both proof obligations are trivial. For attributes, recall $o_{p \boxplus q} = o_{p \boxtimes q} = o_{p \boxast q} = o_p \times o_q$. Thus, to establish, for example (6.14), as far as attributes are concerned, just reason as follows:

$$\begin{aligned}
& \mathsf{a} \cdot o_{(p \boxplus q) \boxplus r} \\
= & \quad \{ \ \boxplus \text{ definition } \} \\
& \mathsf{a} \cdot (o_p \times o_q \times o_r) \\
= & \quad \{ \ \mathsf{a} \text{ natural } \} \\
& (o_p \times (o_q \times o_r)) \cdot \mathsf{a} \\
= & \quad \{ \ \boxplus \text{ definition } \} \\
& o_{p \boxplus (q \boxplus r)} \cdot \mathsf{a}
\end{aligned}$$

On the other hand, preservation of seed predicates resorts to the fact that $\langle \mathbf{2}, \wedge, \mathsf{true} \rangle$ also forms an Abelian monoid. Back to the associativity of $\boxplus$, we proceed as follows:

$$\begin{aligned}
& \gamma_{(p \boxplus q) \boxplus r} \\
= & \quad \{ \ \boxplus \text{ definition } \} \\
& \wedge \cdot (\wedge \times \mathsf{id}) \cdot (\gamma_p \times \gamma_q \times \gamma_r) \\
= & \quad \{ \ \mathsf{a} \text{ isomorphism } \} \\
& \wedge \cdot (\wedge \times \mathsf{id}) \cdot (\gamma_p \times \gamma_q \times \gamma_r) \cdot \mathsf{a}^\circ \cdot \mathsf{a} \\
= & \quad \{ \ \mathsf{a}, \mathsf{a}^\circ \text{ natural } \} \\
& \wedge \cdot (\wedge \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\gamma_p \times (\gamma_q \times \gamma_r)) \cdot \mathsf{a} \\
= & \quad \{ \ \wedge \text{ associative } \} \\
& \wedge \cdot (\mathsf{id} \times \wedge) \cdot (\gamma_p \times (\gamma_q \times \gamma_r)) \cdot \mathsf{a} \\
= & \quad \{ \ \boxplus \text{ definition } \} \\
& \gamma_{p \boxplus (q \boxplus r)} \cdot \mathsf{a}
\end{aligned}$$

3. Interaction

21. The definition of feedback operators intended to capture component interaction depends essentially on the shape of the input-output interface. For *object* components we define again both a (family of) *feedback* and *partial feedback* combinators which leave the interface unchanged. The absence of a direct dependency between input and output in the components' model discussed in this chapter, suggests the introduction of another feedback combinator which, additionally, hides the fed back parameter.

Recall that in §5.78 a similar combination of interaction and restriction was proposed in the context of *separable* components (§5.67). This has been called *hook* as, in a sense, it captures a sort of partial sequential composition.

22. FEEDBACK. Let $p : Z \longrightarrow Z$ be a component. The *feedback* combinator on type Z is defined by

$$p \Lsh = \langle \gamma_p, \langle o_{p \Lsh}, \overline{a}_{p \Lsh} \rangle \rangle$$

where

$$o_{p \Lsh} = o_p$$

and

$$\begin{aligned} a_{p \Lsh} \;=\; & U_p \times Z \xrightarrow{a_p} \mathsf{B} U_p \xrightarrow{\mathsf{B}\langle \mathsf{id}, o_p \rangle} \mathsf{B}(U_p \times Z) \\ & \xrightarrow{\mathsf{B} a_p} \mathsf{B}\mathsf{B} U_p \xrightarrow{\mu} \mathsf{B} U_p \end{aligned}$$

23. LEMMA. Let $f : Z \longrightarrow Z$ be a function, $i : W \longrightarrow Z$ a Set isomorphism, $p : Z \longrightarrow Z$ and $q : T \longrightarrow T$ components. Then,

$$\ulcorner f \urcorner \Lsh \sim \ulcorner f \cdot f \urcorner \tag{6.23}$$

$$p \Lsh [i, i^\circ] \sim p[i, i^\circ] \Lsh \tag{6.24}$$

$$(p \boxplus q) \Lsh \sim p \Lsh \boxplus q \Lsh \tag{6.25}$$

$$(p \boxtimes q) \Lsh \sim p \Lsh \boxtimes q \Lsh \tag{6.26}$$

$$(p \boxast q) \Lsh \sim p \Lsh \boxast q \Lsh \tag{6.27}$$

Proof. [Appendix D, page 400].

□

24. PARTIAL FEEDBACK. The *partial feedback* combinator applies to components p typed as $I + Z \longrightarrow O \times Z$. Component $p \Lsh_Z$ is executed for an arbitrary input. Then the attribute of p is read and its Z component fed back to p. Thus, differently of what happens in the *functional* case (§5.51), the state of p is always updated twice. Formally,

$$p \Lsh_Z = \langle \gamma_p, \langle o_{p \Lsh_Z}, \overline{a}_{p \Lsh_Z} \rangle \rangle$$

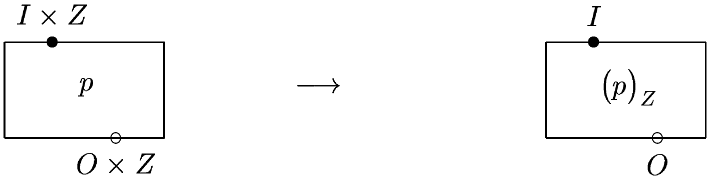
where

$$o_{p\Lsh_Z} = o_p$$

and

$$\begin{aligned} a_{p\Lsh_Z} = {} & U_p \times (I+Z) \xrightarrow{a_p} \mathsf{B}U_p \xrightarrow{\mathsf{B}\langle \mathsf{id},\pi_2 \cdot o_p\rangle} \mathsf{B}(U_p \times Z) \\ & \xrightarrow{\mathsf{B}(\mathsf{id}\times\iota_2)} \mathsf{B}(U_p \times (I+Z)) \\ & \xrightarrow{\mathsf{B}a_p} \mathsf{BB}U_p \xrightarrow{\mu} \mathsf{B}U_p \end{aligned}$$

25. HOOK. As mentioned above, the *hook* combinator applies to cases in which the required input is a pair whose right projection comes from the (observation of the previous) state. Note the hiding of the common parameter. The corresponding diagram is



Formally,

$$(p)_Z = \langle \gamma_p, \langle o_{(p)_Z}, \overline{a}_{(p)_Z} \rangle\rangle$$

where

$$o_{(p)_Z} = \pi_2 \cdot o_p$$

and

$$a_{(p)_Z} = U_p \times I \xrightarrow{\langle \pi_1, \langle \pi_2, \pi_2 \cdot o_p \cdot \pi_1 \rangle\rangle} U_p \times (I \times Z) \xrightarrow{a_p} \mathsf{B}U_p$$

26. As already stressed, the interaction scheme for components is basically a generalisation of *pipelining*. In the component's model discussed in this chapter this can be made quite explicit by showing how to define sequential composition in terms of both *partial feedback* and *hook*. This is proved in the next paragraphs.

27. LEMMA. Let $p : I \longrightarrow K$ and $q : K \longrightarrow O$ be two components. Then

$$(p \boxplus q)[\mathsf{id}, \mathsf{s}] \Lsh_Z [\iota_1, \pi_1] \;\sim\; p \,;\, q \tag{6.28}$$

$$(p \boxast q)[\iota_1, \mathsf{s}] \Lsh_Z [\iota_1, \pi_1] \;\sim\; p \,;\, q \tag{6.29}$$

Proof. [Appendix D, page 401].

□

28. In order to formulate a similar result for the *hook* combinator, we have to introduce a *delay* operator δp which replicates the state space of p so that one copy of U_p always maintains the previous state value. This is the value read by the delayed p attribute. The operator is interesting on its own as it allows for more versatile 'object' composition patterns. This justifies the brief incursion on delays which follows. First the formal definition:

29. DELAY. Given a component $p : I \longrightarrow O$, δp is a component over the same interface defined by

$$\delta p \;=\; \langle \gamma_{\delta p}, \langle o_{\delta p}, \overline{a}_{\delta p} \rangle \rangle$$

where

$$\begin{aligned}
\gamma_{\delta p} &= U_p \times U_p \xrightarrow{\gamma_p \times \gamma_p} \mathbf{2} \times \mathbf{2} \xrightarrow{\wedge} \mathbf{2} \\
o_{\delta p} &= U_p \times U_p \xrightarrow{\pi_1} U_p \xrightarrow{o_p} O
\end{aligned}$$

and

$$\begin{aligned}
a_{\delta p} &= U_p \times U_p \times I \xrightarrow{\pi_2 \times \mathsf{id}} U_p \times I \xrightarrow{\triangle \times \mathsf{id}} U_p \times U_p \times I \\
&\qquad \xrightarrow{\mathsf{a}} U_p \times (U_p \times I) \xrightarrow{\mathsf{id} \times a_p} U_p \times \mathsf{B} U_p \xrightarrow{\tau_l} \mathsf{B}(U_p \times U_p)
\end{aligned}$$

30. LEMMA. For suitably typed components p, q and functions f and g,

$$\delta(p[f, g]) \;\sim\; (\delta p)[f, g] \tag{6.30}$$

$$\delta(p \boxtimes q) \;\sim\; \delta p \boxtimes \delta q \tag{6.31}$$

$$\delta(p \boxplus q) \;\sim\; \delta p \boxplus \delta q \tag{6.32}$$

$$\delta(p \boxast q) \;\sim\; \delta p \boxast \delta q \tag{6.33}$$

Proof. [Appendix D, page 403].

□

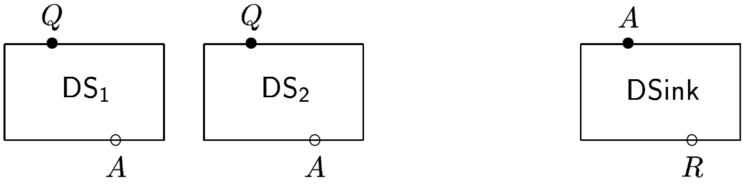
31. LEMMA. It is now possible to state the relationship between *hook* and *sequential composition*. For components $p : I \longrightarrow Z$ and $q : Z \longrightarrow O$, one has

$$\big((p \boxtimes q)[\mathsf{id}, \mathsf{s}]\big)_Z \sim \delta p \,;\, q \tag{6.34}$$

Proof. [Appendix D, page 405].

□

32. EXAMPLE. In order to illustrate the use of the connectives just introduced, consider a distributed querying system in which a question (modeled by a type Q) is simultaneously placed to several independent data sources, each of which supplies a possible answer (of type A). A special component acts as an answer collector, merging all answers produced and computing a final result by means of some pre-defined algorithm. Let us start with two data sources DS_1 and DS_2 (the n-ary case would be dealt similarly), and a concentrator DSink acting as data sink. Note that DS_1 and DS_2 may have different definitions and do not need to be deterministic. A typical choice for B in this example could be the *probabilistic bag* monad mentioned in §4.65 to assign a confidence level to each given answer. Thus,



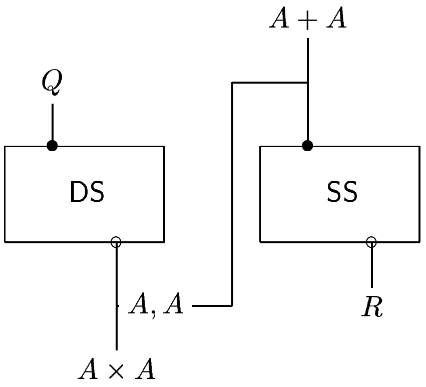
We begin by building the synchronous product $\mathsf{DS}_1 \boxtimes \mathsf{DS}_2$ with input $Q \times Q$ and output $A \times A$. As the same question is to be placed simultaneously to both data sources, input has to be reduced by wrapping this product with the diagonal function, *i.e.*,

$$\mathsf{DS} = (\mathsf{DS}_1 \boxtimes \mathsf{DS}_2)[\triangle, \mathsf{id}]$$

The next step is to compose DS with DSink via $\boxplus$ such that the output of the former is fed back into the latter, through a double application of *partial feedback*. This requires the input of DSink to be expanded to a coproduct of A, an effect which is achieved by wrapping with the appropriate codiagonal

$$\mathsf{SS} = \mathsf{DSink}\,[\nabla, \mathsf{id}]$$

Some additional *wiring* is required to apply the *partial feedback* operator. Because the output interface of $\mathsf{DS} \boxplus \mathsf{SS}$ is $(A \times A) \times R$ it has to be changed to $R \times (A \times A)$ by wrapping with the s isomorphism. Finally, associativity and exchange are needed to prepare for the double partial feedback. The result is pictured in the following diagram, where, in a *naíve* notation, the double feedback of A is represented by an annotation in the connecting line:



Altogether, the whole system is given by

$$\mathsf{S} \;=\; (\mathsf{DS} \boxplus \mathsf{SS})[\mathsf{a}_{+}{}^{\circ}, \mathsf{a}^{\circ} \cdot \mathsf{s}] \Lsh_A \; [\mathsf{xr}_{+}, \mathsf{xr}] \Lsh_A$$

Finally, the intermediate answers are hidden from the environment, yielding the final querying system as component

$$\mathsf{QuS} \;=\; \mathsf{S}[\iota_1, \pi_1]$$

Note in the specification of QuS how *restriction* appears as a particular case of *wrapping*. It should also be remarked that the model for components discussed in the present chapter deals with multiplicative output to additive input connections in a more natural way than the *functional* model discussed previously (see §5.62). This is due, of course, to the explicit separation of input and output.

4. A Note on Internal Activity

33. MOTIVATION. Often the evolution of a software component is driven not only in response to an external stimulus, but also by some kind of internal activity, independent of the component's environment, which the specifier may want to record somehow. For instance, one may be interested in modelling 'noise' or defective behaviour. Or representing a 'maintenance' operation which starts 'silently' from time to time. The incorporation of internal, autonomous activity in component models is thus a point worth to consider. It is not, of course, exclusive to the object component model discussed in this chapter: the basic strategy below applies as well to the 'functional' components addressed in chapter 5.

Suppose the internal activity of a component $p : I \longrightarrow O$ is modeled by an action

$$z : U_p \longrightarrow \mathsf{B}\, U_p$$

A way of expressing the interference of z into the dynamics of p is to redefine the action of p as $\overline{[a_p, z] \cdot \mathsf{dr}} : U_p \longrightarrow (\mathsf{B}U_p)^{I+\mathbf{1}}$. This, however, assumes z typed as $z : U_p \times \mathbf{1} \longrightarrow \mathsf{B}\ U_p$, thus requiring a 'dummy' parameter of type $\mathbf{1}$, which can be regarded as a 'button' which, once triggered, activates z. However, recall that all internal steps are recorded in the component's behaviour. So 'button' pressing would be traceable in the final coalgebra and this, of course, would contradict the very definition of an *internal* action.

A more appropriate solution consists in modifying the action of p so that z is called at each activation of an external action and before it actually takes place. In this way the component interface remains unchanged. Moreover, this conveys the intuitive idea that, when interacting with p the current configuration of its state space may be different from the one left by the former interaction, as a result of some internal activity. We proceed by defining internal action as another operation on components:

34. DEFINITION. Let $p : I \longrightarrow O$ and $z : U_p \longrightarrow \mathsf{B}\ U_p$ denote, respectively, a component and a specified internal action. The *extension* of p with z, written $z \rhd p$, is a component over U_p which satisfies γ_p and is given by

$$\begin{aligned} o_{z \rhd p} &= o_p \\ a_{z \rhd p} &= a_p \bullet \tau_r \cdot (z \times \mathsf{id}) \end{aligned}$$

35. REMARK. Notice that the action of $z \rhd p$ is still a coalgebra for endofunctor (6.1). Moreover, although its behaviour differs from the behaviour of p, *i.e.*, in general $[\![\overline{a}_p]\!]\ u \neq [\![\overline{a}_{z \rhd p}]\!]\ u$, for a valid initial state $u \in U_p$, there are no traces of z being 'triggered' in the corresponding final coalgebra. The following results characterise internal extension.

36. LEMMA. Let p and q be two components, $h : p \longrightarrow q$ a seed predicate preserving comorphism and both $z_1 : U_p \longrightarrow \mathsf{B}\ U_p$ and $z_2 : U_q \longrightarrow \mathsf{B}\ U_q$ internal actions over U_p and U_q. Suppose also that $z_2 \cdot h = \mathsf{B}\ h \cdot z_1$, that is, *$h$ preserves internal activity*. Then h is still a seed's predicate preserving comorphism from $z_1 \rhd p$ to $z_2 \rhd p$.

Proof. All we have to show is that the action parts of the extended components commute with h, *i.e.*,

$$\overline{a}_{z_2 \rhd q} \cdot h \;=\; \mathsf{B}\ h^I \cdot \overline{a}_{z_1 \rhd p} \tag{6.35}$$

Note that, by definition, 'extension' does not interfere neither with the observers' part nor with the seed predicates. Therefore, all that has to be checked is

$$\begin{aligned}
& a_{z_2 \rhd q} \cdot (h \times \mathsf{id}) \\
= \quad & \qquad \{ \ \rhd \text{ definition and } \times \text{ functor} \ \} \\
& a_q \bullet \tau_r \cdot ((z_2 \cdot h) \times \mathsf{id}) \\
= \quad & \qquad \{ \ \text{assumption: } h \text{ comorphism} \ \} \\
& a_q \bullet \tau_r \cdot ((\mathsf{B}\ h \cdot z_1) \times \mathsf{id}) \\
= \quad & \qquad \{ \ \tau_l \text{ natural (C.5) and } \times \text{ functor} \ \} \\
& a_q \bullet \mathsf{B}\ (h \times \mathsf{id}) \cdot \tau_r \cdot (z_1 \times \mathsf{id}) \\
= \quad & \qquad \{ \ \bullet \text{ definition} \ \} \\
& \mu \cdot \mathsf{B}\ a_q \cdot \mathsf{B}\ (h \times \mathsf{id}) \cdot \tau_r \cdot (z_1 \times \mathsf{id}) \\
= \quad & \qquad \{ \ \mathsf{B} \text{ functor} \ \} \\
& \mu \cdot \mathsf{B}\ (a_q \cdot (h \times \mathsf{id})) \cdot \tau_r \cdot (z_1 \times \mathsf{id}) \\
= \quad & \qquad \{ \ \text{assumption: } h \text{ comorphism} \ \} \\
& \mu \cdot \mathsf{B}\ (\mathsf{B}\ h \cdot a_p) \cdot \tau_r \cdot (z_1 \times \mathsf{id}) \\
= \quad & \qquad \{ \ \mathsf{B} \text{ functor} \ \} \\
& \mu \cdot \mathsf{B}\ \mathsf{B}\ h \cdot \mathsf{B}\ a_p \cdot \tau_r \cdot (z_1 \times \mathsf{id}) \\
= \quad & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
& \mathsf{B}\ h \cdot \mu \cdot \mathsf{B}\ a_p \cdot \tau_r \cdot (z_1 \times \mathsf{id}) \\
= \quad & \qquad \{ \ \bullet \text{ definition} \ \} \\
& \mathsf{B}\ h \cdot a_p \bullet \tau_r \cdot (z_1 \times \mathsf{id})
\end{aligned}$$

$$\begin{array}{ll} = & \{ \ \rhd \text{ definition } \} \\ & \mathsf{B}\ h \cdot a_{z_1 \rhd p} \end{array}$$

□

37. LEMMA. Internal extension commutes with *wrapping* and *parallel* composition. It is immediate to check, however, that it does not commute with the remaining combinators. Thus,

$$z \rhd (p[f,g]) \ = \ (z \rhd p)[f,g] \tag{6.36}$$

and, for B commutative,

$$(z \boxtimes w) \rhd (p \boxtimes q) \ = \ (z \rhd p) \boxtimes (w \rhd q) \tag{6.37}$$

where, given internal actions $z : U_p \longrightarrow \mathsf{B}\ U_p$ and $w : U_q \longrightarrow \mathsf{B}\ U_q$, $z \boxtimes w$ is defined by

$$z \boxtimes w \ = \ U_p \times U_q \xrightarrow{z \times w} \mathsf{B}\ U_p \times \mathsf{B}\ U_q \xrightarrow{\delta_l} \mathsf{B}\ (U_p \times U_q)$$

Proof. Concerning equation (6.36) the reasoning is as follows:

$$\begin{array}{ll} & a_{z \rhd (p[f,g])} \\ = & \{ \ \rhd \text{ and } \textit{wrapping} \text{ definitions } \} \\ & \mu \cdot \mathsf{B}a_p \cdot \mathsf{B}(\mathsf{id} \times f) \cdot \tau_r \cdot (z \times \mathsf{id}) \\ = & \{ \ \tau_r \text{ natural (C.5) } \} \\ & \mu \cdot \mathsf{B}a_p \cdot \tau_r \cdot (\mathsf{id} \times f) \cdot (z \times \mathsf{id}) \\ = & \{ \ \times \text{ functor, } \rhd \text{ and } \textit{wrapping} \text{ definitions } \} \\ & a_{(z \rhd p)[f,g]} \end{array}$$

Concerning the other equation,

$$\begin{array}{ll} & a_{(z \boxtimes w) \rhd (p \boxtimes q)} \\ = & \{ \ \rhd \text{ definition } \} \\ & \mu \cdot \mathsf{B}a_{p \boxtimes q} \cdot \tau_r \cdot (\delta_l \times \mathsf{id}) \cdot (z \times w \times \mathsf{id}) \\ = & \{ \ \boxtimes \text{ definition } \} \\ & \mu \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(a_p \times a_q) \cdot \mathsf{Bm} \cdot \tau_r \cdot (\delta_l \times \mathsf{id}) \cdot (z \times w \times \mathsf{id}) \\ = & \{ \ \text{law (C.80) } \} \\ & \mu \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(a_p \times a_q) \cdot \delta_l \cdot (\tau_r \times \tau_r) \cdot \mathsf{m} \cdot (z \times w \times \mathsf{id}) \end{array}$$

$$
\begin{array}{ll}
= & \quad \{\ \delta_l \text{ natural (C.31)}\ \} \\
& \mu \cdot \mathsf{B}\delta_l \cdot \delta_l \cdot (\mathsf{B}a_p \times \mathsf{B}a_q) \cdot (\tau_r \times \tau_r) \cdot \mathsf{m} \cdot (z \times w \times \mathsf{id}) \\
= & \quad \{\ \text{law (C.75), which requires } \mathsf{B} \text{ commutativity}\ \} \\
& \delta_l \cdot (\mu \times \mu) \cdot (\mathsf{B}a_p \times \mathsf{B}a_q) \cdot (\tau_r \times \tau_r) \cdot \mathsf{m} \cdot (z \times w \times \mathsf{id}) \\
= & \quad \{\ \mathsf{m} \text{ natural}\ \} \\
& \delta_l \cdot (\mu \times \mu) \cdot (\mathsf{B}a_p \times \mathsf{B}a_q) \cdot (\tau_r \times \tau_r) \cdot ((z \times \mathsf{id}) \times (w \times \mathsf{id})) \cdot \mathsf{m} \\
= & \quad \{\ \boxtimes \text{ and } \rhd \text{ definitions}\ \} \\
& a_{(z \rhd p) \boxtimes (w \rhd q)}
\end{array}
$$

□

5. Monadic Morphisms

38. This chapter closes with the discussion of a possible extension of the component category St_{B} stemming from a broader definition of component morphism. The motivation is increased flexibility, as it will be shown that monadic functions can be used to *wire* components. As explained below, the additional structure on the wiring function is ‘absorbed’ by the component’s behaviour model.

39. MORPHISMS. A component morphism $h : p \longrightarrow q$ in the new category — which will be denoted by Sm_{B} in the sequel — is a coalgebra morphism up to a natural transformation $\tau_{f,g}$ encoding interface conversion, parametric on functions f on the output and g on the input. Technically, this amounts to a function h between the corresponding state spaces making the following diagram to commute

$$\begin{tikzcd}
U_p \arrow[rr, "h"] \arrow[d, "\overline{p}"'] & & U_q \arrow[d, "\overline{q}"] \\
\mathsf{T}^{\mathsf{B}}_{I,O} U_p \arrow[r, "\tau_{f,g}"] & \mathsf{T}^{\mathsf{B}}_{I',O'} U_p \arrow[r, "\mathsf{T}^{\mathsf{B}}_{I',O'} h"] & \mathsf{T}^{\mathsf{B}}_{I',O'} U_q
\end{tikzcd}$$

where $\mathsf{T}^{\mathsf{B}}_{I,O}$ denotes endofunctor (6.1) with I and O as interface types. Furthermore, h has to preserve seed predicates, as before.

We need, however, to be more explicit on f and g in the definition of $\tau_{f,g}$. For the deterministic case (*i.e.*, $\mathsf{B} = \mathsf{Id}$), $\tau_{f,g}$ is simply function $f \times \mathsf{id}^g$, for $f : O \longrightarrow O'$ and $g : I' \longrightarrow I$. However, the presence of a non trivial (monadic) behaviour model calls for a broader definition. The basic observation is that each function $g : I' \longrightarrow \mathsf{B}I$

induces a function U_g from $(\mathsf{B}U)^I$ to $(\mathsf{B}U)^{I'}$ given by $\_ \bullet g$, where $\bullet$ is the Kleisli composition for monad B.

U_g verifies a suitable rephrasing of the usual properties of exponentials, as detailed below.

40. LEMMA. Function U_g, for $g : I' \longrightarrow \mathsf{B}I$, is natural on U.

Proof. Let $h : U \longrightarrow V$. Then, unfolding the corresponding definitions,

$$\begin{aligned} (\mathsf{B}h)^{I'} \cdot U_g &= (\mathsf{B}h \cdot \_) \cdot (\_ \bullet g) \\ &= \mathsf{B}h \cdot \_ \bullet g \\ &= (\_ \bullet g) \cdot (\mathsf{B}h \cdot \_) \\ &= V_g \cdot (\mathsf{B}h)^I \end{aligned}$$

□

41. Furthermore, $U_\_$ is a contravariant functor from the Kleisli category for B to Set, assigning $(\mathsf{B}U)^I$ to object I and U_g to Kleisli arrow $g : I' \longrightarrow \mathsf{B}I$. Clearly, $U_{\eta_I} = \mathsf{id}_{(\mathsf{B}U)^I}$. On the other hand, $U_{g'} \cdot U_g = U_{g \bullet g'}$ is a direct corollary of the following more general law.

42. LEMMA. For $f : U \longrightarrow U'$ and $g : I' \longrightarrow (\mathsf{B}U)^I$, define $f_g : (\mathsf{B}U)^I \longrightarrow (\mathsf{B}U')^{I'}$ by

$$f_g = (\mathsf{B}f)^{I'} \cdot (\mathsf{B}U)_g = \mathsf{B}f \cdot \_ \bullet g$$

Now, let $f' : U' \longrightarrow U''$ and $g' : I'' \longrightarrow (\mathsf{B}U)^{I'}$. Then,

$$f'_{g'} \cdot f_g = (f' \cdot f)_{g \bullet g'} \tag{6.38}$$

Proof.

$$\begin{aligned} f'_{g'} \cdot f_g &= (\mathsf{B}f' \cdot \_ \bullet g') \cdot f_g \\ &= \mathsf{B}f' \cdot f_g \bullet g' \\ &= \mathsf{B}f' \cdot (\mathsf{B}f \cdot \_ \bullet g) \bullet g' \\ &= (\mathsf{B}f' \cdot f) \cdot \_ \bullet (g \bullet g') \\ &= (f' \cdot f)_{g \bullet g'} \end{aligned}$$

□

43. THE Sm_B CATEGORY. Summing up the previous discussion, a morphism between B-components p and q over U and V, respectively, can be presented as a pair $\langle h, \tau_{f,g} \rangle$, with $f : O \longrightarrow O'$, $g : I' \longrightarrow \mathsf{B}I$, where $h : U \longrightarrow V$ is seed-preserving and makes the diagram in §39 to commute. $\tau_{f,g}$ is the natural transformation whose U component is $(\tau_{f,g})_U \; = \; f \times U_g$ with U_g defined wrt the Kleisli composition for B. Given two such arrows $\langle h, \tau_{f,g} \rangle$ and $\langle h', \tau_{f',g'} \rangle$, their composition is the pair

$$\langle h' \cdot h, \tau_{f' \cdot f, g \bullet g'} \rangle$$

Finally, identities are defined by $\langle \mathsf{id}_U, \tau_{\mathsf{id}_O, \eta_I} \rangle$. Components and component morphisms form, therefore, a category denoted by Sm_B, or simply Sm.

44. A crucial point is to ensure that the proposed definition for $\tau_{f,g}$ subsumes the standard case in which the monadic effect is not present. Note that, for the maybe monad, monadic g is just the classifier (or "totalizer") of a partial map $g' : I' \rightharpoonup I$. If g' is itself total then its classifier g satisfies the equation $g = \iota_1 \cdot g'$ and U_g coincides with $(U + \mathbf{1})^{g'}$. This is indeed the case for the other monads considered above. In fact, a non monadic input morphism always emerges as a special case of a monadic one. That is to say, a total map for the maybe monad, an entire and simple relation for the powerset, and so on. The following lemma proves this fact for the general case.

45. LEMMA Let $g' : I' \longrightarrow I$ and define $g = \eta_I \cdot g'$. Then $U_g = (\mathsf{B}U)^{g'}$.

Proof. Let $\phi : I \longrightarrow \mathsf{B}\, U$. Then

$$\begin{array}{ll}
 & U_g \; \phi \\
= & \quad \{ \text{ definition } \} \\
 & \phi \bullet g \\
= & \quad \{ \; g \text{ is non-monadic } \} \\
 & \phi \bullet (\eta_I \cdot g') \\
= & \quad \{ \; \bullet \text{ definition } \} \\
 & \mu \cdot \mathsf{B}\; \phi \cdot \eta_I \cdot g' \\
= & \quad \{ \; \mu \text{ natural (C.16) } \} \\
 & \mu \cdot \eta_{\mathsf{B}\, U} \cdot \phi \cdot g' \\
= & \quad \{ \text{ law (C.14) } \} \\
 & \phi \cdot g' \\
= & \quad \{ \text{ definition } \}
\end{array}$$

$$(\mathsf{B}\ U)^{g'}\ \phi$$

□

46. WRAPPING REVISITED. We end this section by generalising the *wrapping* combinator introduced in §16 in order to allow for pre-composition of components with monadic arrows. First, however, we need to define a twin category of interface spaces for components.

47. INTERFACES. For each behaviour monad B, the interface category $\mathsf{Intf}_{\mathsf{B}}$ (abbv. Intf), has pairs of sets $\langle O, I\rangle$ as objects and pairs of functions as arrows. In particular, a morphism $l : \langle O, I\rangle \longrightarrow \langle O', I'\rangle$ is defined as a pair $\langle f, g\rangle$, with $f : O \longrightarrow O'$ and $g : I' \longrightarrow \mathsf{B}I$. Composition and identities are pointwise: inherited from Set in the first component and from the Kleisli category for B in the second.

48. The definition of *wrapping* builds on the fundamental observation that Sm is cofibred over Intf, as proved in §49 below. A cofibration is a functor $\mathsf{L} : \mathsf{Sm} \longrightarrow \mathsf{Intf}$ providing co-cartesian liftings of every Intf morphism $l : \langle O, I\rangle \longrightarrow \langle O', I'\rangle$ originated in the L image of each Sm-object. In our context, this means that, given a component p, each interface morphism $l : \mathsf{L}p \longrightarrow \langle O', I'\rangle$ can be lifted (or *extended*) to a component morphism $l_! : p \longrightarrow q$, such that $\mathsf{L}q = \langle O', I'\rangle$ in a canonical way.

Given an interface morphism $l : \langle O, I\rangle \longrightarrow \langle O', I'\rangle$ and a component $p : I \longrightarrow O$, we denote p wrapped by l by $l_!\ p$. In this setting, the basic result is as follows:

49. LEMMA. The functor $\mathsf{L} : \mathsf{Sm} \longrightarrow \mathsf{Intf}$ defined by

$$\begin{aligned} \mathsf{L}(p : I \longrightarrow O) &= \langle O, I\rangle \\ \mathsf{L}\langle h, \tau_{f,g}\rangle &= \langle f, g\rangle \end{aligned}$$

is a cofibration.

Proof. Consider p a component defi ned over U and $\langle f, g\rangle : \mathsf{L}p \longrightarrow \langle O', I'\rangle$ a morphism in Intf (see diagram below). We claim that its cocartesian lifting is $\langle \mathsf{id}_U, \tau_{f,g}\rangle : p \longrightarrow \langle f, g\rangle_!\ p$.

$$\begin{tikzcd}
 & & & t \\
\mathsf{Sm} \arrow[d, "\mathsf{L}"'] & p \arrow[r, "{\langle \mathsf{id}_U, \tau_{f,g} \rangle}"'] \arrow[urr, "{\langle h, \tau_{f',g'} \rangle}"] & \langle f, g \rangle_! \, p \arrow[ur, dotted, "{\langle h, \tau_{f'',g''} \rangle}"'] & \\
\mathsf{Intf} & \langle O, I \rangle \arrow[r, "{\langle f, g \rangle}"] \arrow[drr, "{\langle f', g' \rangle}"'] & \langle O', I' \rangle \arrow[dr, "{\langle f'', g'' \rangle}"] & \\
 & & & \langle O'', I'' \rangle
\end{tikzcd}$$

Let $t : I'' \longrightarrow O''$ be another component over V and $\langle h, \tau_{f',g'} \rangle : p \longrightarrow t$ an Sm-morphism. Suppose that $\langle f', g' \rangle : \langle O, I \rangle \longrightarrow \langle O'', I'' \rangle$ factorizes over $\langle f, g \rangle$ in Intf through $\langle f'', g'' \rangle$. Therefore composition in Intf yields $f'' \cdot f = f'$ and $g \bullet g'' = g'$. We may, then, close the upper diagram, in a unique way, with $\langle h, \tau_{f'',g''} \rangle$. In fact,

$$\begin{aligned}
& \langle h, \tau_{f'',g''} \rangle \cdot \langle \mathsf{id}_U, \tau_{f,g} \rangle \\
= \quad & \{ \text{ definition } \} \\
& \langle h, f'' \times U_{g''} \rangle \cdot \langle \mathsf{id}_U, f \times U_g \rangle \\
= \quad & \{ \text{ lemma §42 } \} \\
& \langle h, (f'' \cdot f) \times U_{g \bullet g''} \rangle \\
= \quad & \{ \text{ factorization in } \mathsf{Intf} \} \\
& \langle h, f' \times U_{g'} \rangle \\
= \quad & \{ \text{ definition } \} \\
& \langle h, \tau_{f',g'} \rangle
\end{aligned}$$

□

CHAPTER 7

Conclusions and Future Work

Summary

This chapter is devoted to a brief discussion and evaluation of the work reported in the thesis and the enumeration of some topics which, in our opinion, deserve further investigation. In particular, some preliminary work on one of these topics — component refinement — is reported.

1. Discussion of Contributions

1. CONTEXT. It is well known that *initial* algebras and *final* coalgebras provide abstract descriptions of a variety of phenomena in programming, in particular of *data* and *behavioural* structures, respectively. Both initiality and finality, as universal properties, entail definitional and proof principles, *i.e.*, a basis for the development of program calculi directly based on (actually driven by) type specifications. Moreover, such properties can be turned into programming *combinators* and used, not only to calculate programs, but also to program with. In functional programming the role of such universals — combined with the 'calculational' style entailed by category theory — has been fundamental to a whole discipline of algorithm derivation and transformation. On the coalgebraic side, *coalgebraic modelling* of dynamical systems has recently emerged as active area of research.

2. SUMMARY OF CONTRIBUTIONS. Framed in this context, the main contribution of this thesis is

- the proposal of a semantic model for software components, regarded as concrete coalgebras for some Set endofunctors, with specified initial conditions, and

- the development of associated component calculi to reason about (and transform) component-based designs.

The notion of a component as addressed in the thesis stems from the context of model oriented specification methods. It is characterised by the presence of an internal state space, which persists in time, and by an interaction model which reflects the asymmetric nature of input and output. Two basic classes of models have been considered, with, respectively, a 'functional' and an 'object-oriented' shape.

The proposed framework distinguishes *components*, as state-based dynamical systems, from their *behaviours*, defined for each component as its anamorphic image (*i.e.*, as elements of the corresponding final coalgebra computed by coinductive extension). Behaviours are, thus, *processes*, as considered in the literature on process algebra. This leads to a second theme addressed in the thesis:

- the 'reconstruction' of classical (*i.e.*, CCS-like) process calculi on top of a coinductive representation of processes and adopting an essentially equational, pointfree calculational style.

In both cases, what distinguishes the work reported in the thesis from similar research documented in the literature is

- the seek for *genericity*, in the sense that the proposed models and calculi for both components and processes are parametric on the *behaviour model* and the *interaction discipline*, respectively.

This has been achieved by introducing parametrization on top of *strong monads*, in the first case, and *positive monoids* of actions, in the other.

Some minor contributions may also be singled out:

- the formulation and proof of a number of *context laws* relating expression 'housekeeping' morphisms with monad multiplication, unit, strength and strength distribution

which we believe has some interest on its own as an 'add-in' to the 'pointfree calculator toolbox', and

- a concern with model *prototyping*, by animation in CHARITY, therefore borrowing to the 'co-side' of computational systems a widely successful design principle.

The following paragraphs discuss these contribuitions in comparison with related approaches, thus placing our evaluation in a broader context.

3. PROCESSES AS CODATA. Looking at processes as elements of a coinductive data type is not new. The original motivation of P. Aczel landmark work [Acz88, Acz93] was precisely the construction of a final semantics for CCS-like languages.

The same line of thought is pursued in, *e.g.*, [RT94, Len98, Wol99, Bal00], among others. References [Len98] and [Bal00], in particular, study models for the π-calculus [MPW92] fully abstract with respect to different notions of observational equivalence.

What may distinguish the approach proposed in chapter 4 of this thesis from the references above, is that our emphasis is placed on the *design*, 'engineering', side. We intended to develop process calculi just as, in the 'inductive half of the world', functional programmers define, say, operations on finite trees and prove their properties. Therefore, we were concerned with genericity, animation and the use of a calculational proof style. Genericity is achieved through the introduction of an *interaction structure*, an elaboration of G. Winskel notion of a synchronisation algebra [WN95], which makes it possible to separate structural and interaction issues in process modelling. Interaction structures are characterised as Abelian positive monoids with a zero element. The monoid structure provides a uniform characterisation of interaction disciplines. In particular, the proposed definition of parallel composition, in terms of synchronous product and interleaving, avoids the need for $\star$, a constant used in [WN95] to denote asynchronous occurrence. How different structural decisions, reflected in the shape of the underlying functor, affect the resulting calculi, was also discussed in this chapter.

4. PROOF STYLE. Throughout chapters 4 to 6 we have adopted a pointfree, essentially equational calculational proof style, which replaces the more traditional use of coinduction (in terms of explicit construction of bisimulations) favoured in coalgebra circles. In chapter 4 a 'conditional fusion' theorem has been proved to handle the derivation of conditional laws.

In a sense, such a proof style is a price to be paid for *genericity*, as properties are to be verified without fixing the working functor completely. Perhaps one might have arrived at suitable notions of 'generic' bisimulation in which coinductive proofs could have been based. Recognising that we have not followed such a direction, and being unaware of any work in the area, we suspect that the formal complexity involved would be comparable to the one specific to the approach adopted here.

Generic proofs performed in this style are often long, even if easy to follow. In most cases their length results from the systematic recording of almost all elementary steps. On the other hand, this style has become familiar to the functional programming community, where it has been popularised under the 'Bird-Meertens formalism' heading (see *e.g.*, [Bac88, Fok92a, BM97] or [BJJM98]).

5. BICATEGORIES OF COMPONENTS. The reason why *components* are fundamentally different from *processes* has already been stressed: they cannot be reduced to purely behavioural structures, as the state space cannot be abstracted away. Each component specification introduces internal dynamics (captured by a concrete coalgebra) defined in terms of external interaction, internal 'memory' (or state) and a general behaviour model.

The bicategorical setting adopted in chapter 5 (and, to a lesser extent, in chapter 6), seemed appropriate to capture a 'two-level structure' in the component models. This is clearly in debt to previous work by R. Walters and his collaborators on models for deterministic input-driven systems [KSW97a, Kat96, KSW00], briefly mentioned in §1.7. Let us mention what distinguishes the approach proposed in this thesis:

- *Genericity.* R. Walters' work deals essentially with deterministic systems (on the grounds that nondeterminism can be captured eventually on suitable categories of behaviours). Our parametrization of components over a strong monad, specifying a behaviour model, is, to the best of our knowledge, original. Such a monadic parametrization allows one to focus on the relevant structure of components, factoring out details about the specific behavioural effects that may be produced. It also enforces a (still very elementary) classification of behaviour models by the set of laws they satisfy (recall the discussion in chapter 5 of a number of laws requiring, for example, monad commutativity or the absence of termination).
- *Interconnection patterns.* A beautiful topic in Walters' work is the way process' bicategories arise from a monoidal category $\langle \mathsf{M}, \otimes \rangle$, by a standard, so-called *loops-suspension* construction. Processes $\langle U, \alpha \rangle : I \longrightarrow O$ are defined in terms of M-morphisms

  $$\alpha : I \otimes U \longrightarrow U \otimes O$$

  where U is the internal state space and I, O correspond to the interface types. In [KSW97a], M is successively taken as $\langle \mathsf{Set}, \times \rangle$ and $\langle \mathsf{Set}, + \rangle$. The result, in the first case, is a category of so-called 'circuits' (or, in our terminology, 'deterministic functional components'). In the other case, input I is identified with a set of initial states (rather than with action stimuli) and output O with final, or 'equilibrium', states. This emphasises state decomposition, through sums, leading to a model of (deterministic) 'while programs'.

  For arbitrary M, the resulting algebra is 'driven' by the original tensor $\otimes$ in M, reducing itself to two aggregation operations, corresponding to 'series' and 'parallel' composition (*i.e.*, our **;** and $\boxtimes$, for $\mathsf{B} = \mathsf{Id}$), and a feedback operator to be discussed below. Such an algebra seems enough to characterise binary circuits (as extensively done in [KSWW01]) and, one may argue, its somehow restricted expressive power is a corollary of the very general construction adopted. In this thesis, however, we have found it necessary to use the full (distributive) structure of Set. This has led to a richer ontology of interconnection patterns for components. In particular, we have considered a *choice* combinator, *wrapping* and *concurrent* composition. We have also investigated the existence of universals, notably, *initial* and *final* objects as well as *either* and *split* constructions, often characterised in weak or conditional forms for different classes of possible behaviour monads.

- *Feedback*. Feedbacks as considered in R. Walters' approach are reminiscent of the corresponding notion in circuit design formalisms (notably in [JS90]). They are essentially *trace* operators in the sense of [JSV96][1]. The feedback combinator in [KSW97a] is defined under the following assumption: given $\langle U, \alpha \rangle : I \otimes Z \longrightarrow Z \otimes O$, Z can be fed back into the system, providing there exists a (essentially unique) 'structure map' $\tau : I \otimes U \longrightarrow Z$. This conveys intuition on feedback as an operator whose applicability depends on the ability to produce Z elements from U and I. The definition, while satisfying the axioms for a trace, seems a bit contrived when taking Set as the underlying category. The 'natural' way to define τ seems to be through a second application of α, assuming a 'seed' value of type Z. This is basically what is done in [KSW97a].

  We have followed a similar idea, but opted for introducing feedback combinators directly, dispensing with explicit 'structure maps'. A basic concern has been, in particular, to distinguish total from partial input feedback as both situations seemed useful in practice. Moreover — and this is a main departure from [KSW97a] — the feedback parameters are not hidden from the component interface. In fact, together with sequential composition, feedback (applied to $\boxplus$, $\boxtimes$ or $\boxast$ composites) is our basic mechanism for component interaction. Again we have found that sticking to the well-established process algebra principle of separating *hiding* (or restriction) and *communication* as different operators, provided crucial increased flexibility. Therefore, our feedback operators have a broader scope of application but, as one could expect, the absence of hiding and the need to deal with both additive and multiplicative interfaces, entail, in the general case, less expressive theories. In particular, they fail to verify the trace axioms.

  In general, trace combinators seem not to possess a simple, 'natural', definition in Set. This is in contrast with what happens in the category Rel

[1] Broadly speaking, a *trace* is a combinator mapping arrows $f : I \otimes Z \longrightarrow Z \otimes O$ into $\mathrm{Tr}\, f : I \longrightarrow O$ and satisfying a number of axioms. Trace combinators are closely related to the existence of fixed points in the underlying categories, having been studied in several contexts, notably in categorical models for linear logic (see, eg, [AJ94]) and process models (not only in Walters' work but also in, *e.g.*, [Sel99]). Reference [BCS98] is a comprehensive account of (local) trace theory, the qualification 'local' referring to traces being defined only for some objects of a category.

of sets and relations (and, in general, in any compact closed category). The canonical notion is, in such a setting, crisp and clear[2]: given a relation $r : I \otimes Z \longrightarrow Z \otimes O$ $\mathrm{Tr}\, r : I \longrightarrow O$ is defined as $\langle i, o \rangle \in \mathrm{Tr}\, r$ iff there is a $z \in Z$ such that $\langle \langle i, z \rangle, \langle z, o \rangle \rangle \in r$. A similar, elegant, notion of feedback arises in the bicategory induced by $\langle \mathsf{Rel}, \otimes \rangle$ used in [KSWW01] to model asynchronous circuits. Modelling feedback by a trace combinator seems, however, to be somehow restrictive, in a number of situations. In [BCS98], for example, it is remarked, for example, that the usual notion of feedback in 'stream processing' fails to satisfy the main non-structural trace axiom — that of 'yanking' — which states that feedback on the $\otimes$ commutativity isomorphism is the identity. Computationally, this is better identified with a 'delay': the output is delayed until it is used as input in the 'next time step'. To fix this mismatch, [KSW00] proposes a more general feedback operator which simply makes the fed back parameter part of the internal state space. Formally, given a system $\langle U, \alpha \rangle : I \otimes Z \longrightarrow Z \otimes O$, the fed back system is given by

$$\langle U \otimes Z, (\mathsf{s} \otimes \mathsf{id}_O) \cdot \alpha \rangle : I \longrightarrow O$$

The notion is appealing, not only because traces can be recovered as special cases (in which feedback is 'instantaneous'), but also because it seems closer to the intuitive notion of a 'feedback', rather than our own 'two-step-in-one' characterisation. Besides a possible difficulty in making sense of 'action internalising' in the presence of additive interfaces, and our decision of keeping *interaction* and *hiding* separate, adoption of a similar construction in the component's calculi would also require the specification of new, extra seed values.

- *Behaviour.* Central to R. Walters' approach is the definition of functors to specified semantic categories, corresponding to different ('application-suited') notions of behaviour. Rel is a possible choice, for very elementary circuits. Another possibility (used, namely, in [KSWW01]) is the category whose objects are infinite streams of input-state-output values. We have, however, sticked to a canonical notion of behaviour: that computed by coinductive extension. This has the advantage of being uniformly defined, providing a simple view of behaviours as 'codata' which is easier to reason about and to animate in a functional programming framework. In

[2] Our claim about the difficulty of defining traces on $\langle \mathsf{Set}, \times \rangle$ is easily illustrated by specialising the construction which follows to relations arising as function graphs. Please notice that tensor $\otimes$ in Rel is the Cartesian product on objects and defined, on relations r and t, by $\langle \langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle \rangle \in r \otimes t$ iff $\langle x_1, y_1 \rangle \in r$ and $\langle x_2, y_2 \rangle \in t$.

practice, however, we may have the need to work with notions of observation equivalence which, being less discriminant than bisimilarity, are also of a 'less structural' nature. A possibility that has been studied elsewhere (namely in the context of interaction categories [AGN94] and, in particular, in D. Spooner thesis [Spo97]) is to single out particular classes of 2-cells in which such equivalences could be based.

Finally, two other influences to the approach to component modelling proposed in the thesis should be mentioned. The first is the recent area of coalgebraic specification of object-oriented systems (see *e.g.*, [Rei95, Jac96b]), which has been developed with a similar motivation, although in a property-oriented, or axiomatic, framework. The other is the 'dataflow paradigm' [Oli84] to which some of the aggregation patterns and the general idea of structured wiring can eventually be traced back.

6. PROTOTYPING. We have already insisted on the relevance of having simple ways to animate processes and, more generally, components' behaviours. The direct correspondence between the programs developed and the formal definitions, as well as the straightforward way in which a process interpreter, parametric on the interaction discipline, has been implemented is certainly a merit of CHARITY [CF92]. In fact, its choice as a prototyping language seems to have been the right one. The main reason for this is the fact that CHARITY supports, with the 'right' semantics, both inductive and coinductive data types, without collapsing them, and enforces a strict discipline for their use. Attempting the same kind of task in a *partial* functional language, such as HASKELL, even resorting to laziness, would require somewhat more contrived encodings.

The idea of using CHARITY to encode transition systems as higher-order types was suggested in [Sch97]. Our own case study on component animation in CHARITY is discussed in [Bar99]. The thesis contribution in this area is limited to the full development of the component algebras, reported in chapters 5 and 6, and of an interpreter for a CCS-like process language, parametrized by the interaction structure. The latter is developed (in chapter 4) as an *apomorphism* [VU97], following a (final) semantics-driven approach and resorting to *strength* to represent an environment of (mutually dependent) process definitions.

7. APPLICATIONS TO SOFTWARE ENGINEERING. This thesis found its earlier motivation in a notion of component arising in the context of *model-oriented* specification methods (as in [Jon86, Spi92] or, more specifically, in [ABNO97]). Eventually, it evolved towards a more general approach which we believe may be useful in starting a coalgebraic study of software components in the broader sense discussed in [WW99]. This includes a relatively rich component interconnection calculus, parametric on the envisaged notion of behaviour, which may have some potential to support the definition of suitable formalisms to specify *software architectures*, a related and more and more relevant research topic. In particular, in dealing with heterogeneous components (*e.g.*, based on different behaviour models) one would simply resort to natural transformations induced by monad morphisms, as briefly explained in the last section of chapter 4.

To a large extent, however, this is future work. In particular, we suspect that the suitability of our approach to specify software architectures or real, 'hard' coordination problems, depends on the development of a notion of *component customising* which, having not been addressed in the thesis, is currently among our research interests (see §13).

2. Future Work

8. We finally discuss some topics for future research. One of them — *refinement* — will be discussed in a more detailed way in a separate section.

9. PROCESS CALCULI. Is it possible to extend the framework developed in chapter 4 to deal with more elaborated process calculi? For example, will *mobility* fit in it? These questions deserve further investigation. In a sense, a positive answer requires the development of coalgebraic models fully abstract with respect to the equivalence notions one might be interested in. Final semantics to mobile calculi, dealing with variable binding and dynamically generated names and localities, seems however hard to obtain. The proposal of [Len98], for example, does not provide compositional interpretations for process combinators of the π-calculus. The same difficulty is experienced in the context of interaction categories [Abr94]. Interpretations for combinators, however, are given in the model proposed in [Bal00] which resorts to two different stages: a 'ground model' fully abstract with respect to a form of ground bisimilarity, and a full model built on top of the latter and fully abstract with respect to the congruence derived from ground bisimilarity. How simple calculation is within such models and whether they can be parametrized by different 'mobile' interaction disciplines, and suitably prototyped, remains a topic for future work. Unpublished work by J. Valença on *dependent streams* [Val00] may be worth considering to capture the intuition that process interaction capacities may change upon receiving a stimulus.

10. EXPLORING THE CO-SIDE. The categorical approach to data types entails generic recursion (and co-recursion) schemes encoded as combinators which are polymorphic on the functor capturing the signature of the type. The most fundamental of such schemes are well known. On the inductive side, *iteration* (also known as *structural recursion*) is captured by *catamorphisms*, while *paramorphisms* express the full power of *primitive recursion*. Dually, on the coinductive side, *coiteration* corresponds to *anamorphisms* and the more general notion of *primitive corecursion* is captured by *apomorphisms*.

Combinators on the coinductive side, heavily used in chapter 4, have been largely overlooked in the literature. This is perhaps because (generic) programming with codata is still in its infancy. One feels challenged, however, to further explore this area, seeking for the duals of several derived combinators which have proved useful in the inductive half of the programming landscape. The simplest example we can think of is the *strong* version of a catamorphism discussed in §3.61 (see [Par00] for a recent investigation of this functional). Given functor T, this is a morphism

$$\mathsf{fold}_{\mathsf{T}}\, \phi \;:\; \mu_{\mathsf{T}} \times C \;\longrightarrow\; U$$

uniquely induced by a structure $\phi : \mathsf{T}\ U \times C \longrightarrow U$. Intuitively, it captures circular definitions of functions depending on an inductive argument and a 'context' C. Reversing the arrows, we would expect to get a *parametric unfold*, typed as

$$\mathsf{punfold}_{\mathsf{T}}\, \phi \;:\; U \;\longrightarrow \nu_{\mathsf{T}} + C$$

uniquely induced by a structure $\phi : U \longrightarrow \mathsf{T}\ U + C$. Clearly, such a morphism cannot be unique in Set. However, $\mathsf{punfold}_{\mathsf{T}}\, \phi$ seems to capture an interesting operation scheme: 'if normal computation fails, yield the context value as the whole response', with possible applications in modelling, *e.g.*, 'roll back' operations. Seeking for categories in which such schemes can be meaningfully expressed and studying their applicability is part of a very recent research collaboration with V. Vene.

11. COMPONENT CLASSES. As we have mentioned before, component calculi developed in chapters 5 and 6 are concerned with component interconnection and interaction, which are 'blind' with respect to their internal functional specification. From an engineering point of view, a challenging direction for future work is the study of *specific* classes of components. Such classes can be identified in any of the two general models proposed by singling out a number of properties. The only example addressed in the thesis was that of *separable* components. Note, by the way, that feedback for this class of components satisfies a richer set of laws than it does in the general case. On may ask to what extent further restrictions would recover (in a meaningful context) the feedback combinator of [KSW00] or even a trace operator.

Another class that seems interesting to explore concerns components whose interface can be split into sets of actions affecting disjoint regions of the state space.

Actions from different regions can be executed in parallel as non interference is guaranteed [Oli97]. In such a class of components, additive ($\boxplus$) interfaces can be replaced by concurrent ($\boxast$) ones.

Reasoning about components involves both laws of the underlying calculus and the properties of their internal specification. Component classes encapsulate the latter. The 'voting system' decomposition, presented in chapter 5, is an example of a calculation relying on an additional assumption about the internal specification — that of separability. Unfortunately, we did not have enough time to go further in that direction, which requires thorough *experimental* research driven by suitable case studies.

12. PROTOTYPING. Future work, in this area, includes the full development of prototyping kernels for both process and component calculi, eventually with a graphical interface. On the pragmatic side, the real challenge here is the design of a (complete) diagrammatic notation to express components' interconnection, as mentioned in §5.64. Whether such a prototyping kernel can be plugged into a verification system — like, *e.g.*, a symbolic model checker — remains an open challenge.

13. CUSTOMISING. This thesis regards components as state-based entities interacting through well defined interfaces of observers and actions. It is often the case, however, that a particular component is *used* in a restricted way, namely as part of a broader system. This entails the need for a specification of the intended behaviour, which is not intrinsic to the component itself, but to its *role* (use) in a particular situation. For example, one may want to prescribe that action a is the initial action or that an action b is to follow each occurrence of a.

We would like to extend our model to deal with such situations, a procedure which may be referred to as component *customising* and bears relevance to the scaling up of our framework to the *software architecture* level. Such a distinction is totally absent from model-oriented specification methods, often leading to undesirable over specification. In process calculi, on the other hand, it may be traced back to Milner's distinction between *static* and *dynamic* process connectives, the later being understood as the source of temporal extension. In CCS, 'prefixing' is the typical example of a dynamic connective. Our component algebra lacks such an operator, as we are dealing with concrete coalgebras instead of pure behaviours. Notice, on the other hand, that 'choice', which is also a dynamic operator in process calculi, is treated, at component level, as an aggregation combinator.

The way we envisage to address this topic is as follows. Suppose one wants to customise the use of a component p. First, a decision on what *using a specification* means has to be made. A simple possibility is just a precedence relation between actions. We believe that such an elementary definition will cover several cases actually arising in practice. Once this has been defined, p suffers a 'state extension'-like (§5.69) operation to incorporate 'historical' information. This may be just a record of the last operation executed, or a complete log file for p. Finally, the *use specification* is provided as context information to the extended p component. In each step such information is confronted to the 'historical' record to validate a possible interaction. The resulting behaviour is computed by a strong anamorphism. As illustrated in chapter 4, when developing the process interpreter, context is directly supported in CHARITY via *strength*, which makes the overall idea easy to implement. Conceptually, note that component *customising* re-introduces in the calculus (a form of) temporal extension.

14. REFINEMENT. Last but not least, the definition of appropriate notions of component *refinement* is perhaps the main topic to be addressed in future work. A refinement theory studies *changes in the representation* of a system, entailing a notion of *substitution*, but not necessarily *equivalence*. This means that the usage of a system according to its specification is still valid if it is actually built according to the (valid) implementation. What is commonly understood by being a *valid usage* is that the corresponding observable consequences are still the same, or, in a less strict sense, a subset thereof. In the literature on concurrent systems there is, however, no general consensus on what a good notion of refinement is.

Some work has already been carried out in this direction. However, its very preliminary character prevents a full presentation in the thesis. Due to the relevance of the topic, we have decided to include part of this (sketchy) material in the next (and last!) section, as a basis for future work.

3. Refinement

15. As mentioned in §14, *refinement* can be defined, in broad terms, as a *transformation* of an 'abstract' into a more 'concrete' design, entailing a notion of *substitution*. There is, however, a diversity of ways of understanding both what *substitution* means, and what such a *transformation* should seek for. Let us briefly mention a number of them:

- In VDM [Jon80] *data refinement* is the process of transforming abstract data structures into more concrete ones, a transformation which presumably entails efficiency (*e.g.*, the conversion of an inductive data type into a 'pointer'-based representation). The refinement of a model (ie, a state space and a set of operations upon it) A into another model B has to fulfil a number of requirements. First of all, the existence of *enough redundancy* in the state space of B to represent all the elements of A is required. This is called in [Jon80] the *adequacy* requirement and is captured by the definition of a surjection from B to A, called the *retrieve function*. Next, *substitution* is regarded as 'complete' in the sense that the (concrete) operations over B accept all the input values accepted by the corresponding abstract ones, and, for the same inputs, the results produced are also the same, up to the retrieve map. If models are specified, as they usually are in VDM, by pre and post-conditions, this amounts to say that, under refinement, neither preconditions are strengthened, nor post-conditions are weakened. Note this approach to data refinement, which can be traced back to Hoare's landmark paper [Hoa72], is consensual among *model-oriented* design methods, even though several variants and alternatives have been proposed in the literature (see [RE98] for a recent account).

- The SETS calculus [Oli90, Oli92b], which is a calculus of data representations, is grounded on identical principles: a refinement is witnessed by a (split) epi, which may induce a *representation invariant* on the concrete side. Each concrete operation is then *calculated* (rather than 'conjectured and verified' as in VDM) by solving the corresponding refinement diagram.
- Most calculation within *process algebras* is carried in terms of *equivalence* relations — see, for example, the long 'implementation' case studies in [Mil89] or [Fen96]. However, several 'observation' preorders have also been proposed in the literature which entail different notions of refinement (see, for example, [AV95] or [FE00]). A typical transformation, captured by most of such preorders, is *reduction of nondeterminism*.
- A totally different point of view is that of *action refinement* [Ace92], in which what was considered an atomic action at the abstract level is 'replaced' by a process over an alphabet of more fine-grain actions. The idea, which seems to have some connections with *transition* decomposition in the area of data bases, is difficult to integrate with the *interleaving* semantics adopted in the most popular process calculi, as it interferes with compositionality.
- Another meaning for the word 'refinement', which also appears in the process algebra literature and, mainly, in the context of object-orientation, is *extension of the functionality*. This means that the concrete model may have new actions added, a procedure we would prefer to call *design sophistication* rather than *refinement*. In general, a preorder capturing functionality extension fails to be a pre-congruence (with respect to typical process combinators). Such is the case of, *e.g.*, the attempt of [WD96] to apply to process refinement the notion of Z *forward simulation*.

16. WHAT IS COMPONENT REFINEMENT? Component refinement can be addressed at three different levels (at least):

- The *interface* level, which is concerned with what one may call *plugging compatibility*. The question is whether a component can be transformed, by suitable wiring, to replace another component with a different interface. Note that, once in our models the structure of the interface types encodes the available operations, this may capture situations of *extension of component's functionality*.
- The *behaviour* level, where the notion of refinement is based on a simulation preorder for the behaviour model specified by monad B. As discussed below, a number of situations may be captured, depending on B, on the simulation preorder adopted and on the refinement relation induced. Nondeterminism reduction is just one possibility among many others. Note that behaviour refinement preserves the component interface, *i.e.*, it takes place inside Cp_{B} hom categories.
- The *data* level, which amounts to the static refinement of the data structure which specifies the component state space. This also preserves component interfaces and, as far as behavioural issues are concerned, entails bisimilarity. Therefore, it will not be considered in the sequel.

The following paragraphs discuss some preliminary ideas on refinement, to foster future study. We shall restrict ourselves to the component model discussed in chapter 5.

17. INTERFACE REFINEMENT. Consider law (5.21) in chapter 5:

$$p \boxplus q \ \sim \ (q \boxplus p)[\mathsf{s}_+, \mathsf{s}_+] \tag{7.1}$$

which states that $p \boxplus q$ and $q \boxplus p$ are bisimilar *up to isomorphic wiring*. This means that the observational effect of component $p \boxplus q$ can be achieved by $q \boxplus p$, providing the interface of the latter is converted to the interface of the former. Such a conversion is achieved by composition with the appropriate *wires*, leading to a notion of *replaceability*.

18. DEFINITION. Let p and q be components. We say that $p : I \longrightarrow O$ is *replaceable* by $q : I' \longrightarrow O'$, or q is a *replacement* of p, and write $p \lessdot q$ if there exist functions $w_1 : I' \longrightarrow I$ and $w_2 : O \longrightarrow O'$, to be referred to as the replacement *witnesses*, such that

$$p \ \sim \ q[w_1, w_2] \tag{7.2}$$

Should w_1 and w_2 be isomorphisms, components p and q are called *interchangeable*, a fact recorded as $p \doteqdot q$.

19. Clearly, $p \boxplus q \doteqdot q \boxplus p$, using isomorphism s_+ as a wire in both cases. In general, components p and q are *interchangeable* if each of them is a replacement of the other. Formally,

$$p \doteqdot q \quad \text{iff} \quad p \lessdot q \ \wedge \ q \lessdot p \tag{7.3}$$

Proof. From left to right note that if fact $p \lessdot q$ is witnessed by isomorphisms w_1 and w_2, one also has $p[w_1^\circ, w_2^\circ] \sim q$, *i.e.*, $q \lessdot p$. In the other direction, let $p \lessdot q$ and $q \lessdot p$ hold, *i.e.*, there exist w_1 and w_2 such that $p \sim q[w_1, w_2]$ and $q \sim p[w_3, w_4]$. This implies $q[w_1, w_2][w_3, w_4] \sim q$ which, by law (5.9) equivales $q[w_1 \cdot w_3, w_4 \cdot w_2] \sim q$. From this one draws $w_1 \cdot w_3 = \mathsf{id}_I$ and $w_4 \cdot w_2 = \mathsf{id}_{O'}$. By a similar argument, using $p[w_3, w_4][w_1, w_2] \sim p$, one gets $w_3 \cdot w_1 = \mathsf{id}_{I'}$ and $w_2 \cdot w_4 = \mathsf{id}_O$. Therefore, w_1, w_2, w_3 and w_4 are isomorphisms and $p \doteqdot q$.

□

20. LEMMA. Replaceability ($\lessdot$) is a partial order up to interchangeablity ($\doteqdot$), which is an equivalence on components.

Proof. Anti-symmetry up to $\doteqdot$ of $\lessdot$ has just been proved (§19). Clearly, $\lessdot$ is reflexive because $p \lessdot p$ is witnessed by $p \sim p[\mathsf{id}, \mathsf{id}]$. On the other hand, if $p \lessdot q$ and $q \lessdot r$ hold, there exist w_1, w_2, w_3 and w_4 such that $p \sim q[w_1, w_2]$ and $q \sim r[w_3, w_4]$. Thus, law (5.9) and transitivity of $\sim$ entails $p \sim r[w_1 \cdot w_3, w_4 \cdot w_2]$, *i.e.*, $p \lessdot r$. Reflexivity and transitivity of $\doteqdot$ are proved similarly and $q \doteqdot p$ is witnessed by the converse of isomorphism witnessing $p \doteqdot q$.

□

21. EXAMPLES. Using $\lessdot$ and $\doteqdot$, the component laws in chapter 5 can be presented in a 'wiring free' form. For example, the set of laws in §5.26 becomes

$$\begin{aligned} (p \boxplus q) \boxplus r \ &\doteqdot p \boxplus (q \boxplus r) \\ \mathsf{nil} \boxplus p \ &\doteqdot p \ \doteqdot \ p \boxplus \mathsf{nil} \\ p \boxplus q \ &\doteqdot q \boxplus p \end{aligned}$$

On the other hand, law (5.53)

$$\ulcorner \iota_1 \urcorner \,;\, (p \boxast q) \sim (p \boxplus q) \,;\, \ulcorner \iota_1 \urcorner$$

gives rise to two replacement inequations:

$$\begin{aligned} \ulcorner \iota_1 \urcorner \,;\, (p \boxast q) \ &\lessdot \ p \boxplus q \\ (p \boxplus q) \,;\, \ulcorner \iota_1 \urcorner \ &\lessdot \ p \boxast q \end{aligned}$$

Also law (5.13) can be neatly interpreted as the statement that every component p can replace an *inert* component, *i.e.*,

$$\mathsf{inert}_{\mathbf{1}} \lessdot p$$

Finally, note that definition §18 does not require every replacement situation to be witnessed by functions whose lifting to Cp_{B} are wires — arbitrary functions, with the right types, can also be used. In general, law (5.10) justifies the following fact:

$$\ulcorner f \urcorner \,;\, p \,;\, \ulcorner g \urcorner \;\lessdot\; p \tag{7.4}$$

22. Relation $\lessdot$, however, fails to be a pre-congruence with respect to the component operators introduced in chapter 5. It is immediate to check that $\boxplus$, $\boxtimes$ and $\boxast$, as well as *wrapping* are preserved, *i.e.*, if $p \lessdot p'$ then, for any q, f and g, $p[f,g] \lessdot p'[f,g]$, $p \boxplus q \lessdot p' \boxplus q$ and, similarly, for the other two tensors. But things are different with respect to sequential composition and feedback. In these cases, the replaced expression may even become wrongly typed: suppose, for example, that $p \lessdot p'$, where $p : Z \longrightarrow Z$ and $p' : Z + I \longrightarrow Z \times O$. Clearly, expression $p' \Lsh$ cannot be formed.

What $p \lessdot p'$ means is that component p can be replaced in *any* context by $p'[w_1, w_2]$, for any functions w_1, w_2 witnessing the fact. The explicit reference to them is actually required, something which is not completely satisfactory in a refinement situation, although expected, as mentioned in §15.

23. BEHAVIOUR REFINEMENT. As mentioned in §15, a component p *behaviouraly* refines component q if the behaviour patterns observed for p are a structural restriction, with respect to monad B, of those of q. Let us attempt to make this 'definition' precise, noting that even the word 'restriction' may be object of different interpretations.

In data refinement, there is a 'recipe' to identify a refinement situation: look for a *retrieve function* to witness it. I.e., a morphism in the relevant category, from the 'concrete' to the 'abstract' model such that the latter can be *recovered* from the former up to a suitable notion of equivalence, though, typically, not in a unique way. In SETS [Oli90] such a retrieve function is an epi and the 'suitable notion of equivalence' is, of course, Set isomorphism.

In our components' framework, however, things do not work this way. The reason is obvious: component morphisms (*i.e.*, 2-cells in Cp_{B}) are (seed preserving) comorphisms which, therefore, entail bisimilarity (recall §3.24). We have to look for some *weaker* notion of a morphism between coalgebras. To proceed in that direction with full genericity would require to replace Set by an order-enriched category as the underlying category. This is done in [TR98] leading to a characterisation of *ordered* bisimulation and a corresponding full abstraction result (which states that the inequality relation on the final coalgebra lifts to a coalgebraic bisimulation). The approach presented in the following paragraphs attempts a *naíve* characterisation of behaviour refinement in the restricted setting of the components model discussed in chapter 5.

24. TRANSITIONS. Let T be an extended polynomial Set endofunctor (§3.10). Recall from §4.3 that any coalgebra $\langle U, p : U \longrightarrow \mathsf{T}U\rangle$ specifies a (T-shaped) transition structure over its carrier U, defined in terms of the *structural membership* relation $\in_{\mathsf{T}} \subseteq U \times \mathsf{T}\, U$:

$$u \longrightarrow_p u' \quad \text{iff} \quad u' \in_{\mathsf{T}} p\, u$$

Structural membership is defined in §4.3. Notice that, given $x \in U$, $X \in \mathsf{T}U$ and a function $h : U \longrightarrow V$, if $x \in_{\mathsf{T}} X$ then $h\ x \in_{\mathsf{T}} \mathsf{T}h\ X$. This fact can be established by induction on the polynomial structure, resorting to the definition of $\in_{\mathsf{T}}$ and functoriality.

Similarly, the dynamics of a component $p : I \longrightarrow O$, based on $\mathsf{T}^{\mathsf{B}} = \mathsf{B}(\mathsf{Id} \times O)^I$, can be expressed in terms of the following transition relation:

$$u \xrightarrow{\langle i,o\rangle}_p u' \quad \text{iff} \quad \langle u', o\rangle \in_{\mathsf{B}} (pu)\ i$$

Now consider two components p and q. Recall that a component morphism from p to q is a seed preserving function $h : U_p \longrightarrow U_q$ such that

$$\mathsf{T}^{\mathsf{B}}\ h \cdot \overline{a}_p \ = \ \overline{a}_q \cdot h \tag{7.5}$$

which we usually rewrite as

$$\mathsf{B}(h \times \mathsf{id}) \cdot a_p \ = \ a_q \cdot (h \times id) \tag{7.6}$$

In terms of transitions, equation (7.6) is translated into the following two requirements (by a straightforward generalisation of an argument in [Rut96]):

$$u \xrightarrow{\langle i,o\rangle}_p u' \ \Rightarrow \ h\, u \xrightarrow{\langle i,o\rangle}_q h\, u' \tag{7.7}$$

and

$$h\, u \xrightarrow{\langle i,o\rangle}_q v' \ \Rightarrow \ \exists_{u' \in U} .\ u \xrightarrow{\langle i,o\rangle}_p u' \ \wedge \ u' = h\ v' \tag{7.8}$$

which capture the fact that, not only p dynamics, as represented by the induced transition relation, is *preserved* by h (7.7), but also q dynamics is *reflected* back over the same h (7.8).

25. A possible way of regarding a component p as being a behavioural refinement of another component q is to consider that p transitions are simply preserved in q. For nondeterministic components, this could be understood simply as set inclusion.

But one may also want to force additional restrictions. For example, to stipulate that if p has no transitions from a given state, q should also have no transitions from the corresponding state(s). Or one may adopt an alternative point of view and become interested not in transition preservation, but in their reflection instead. In an attempt to formalise these ideas, we begin by defining the notion of a *simulation*. This will then be taken as a parameter in the definition of both *forward* and *backward* component morphisms. Forward (respectively, backward) morphisms are shown to preserve (respectively, reflect) their source component dynamics. How 'restrictive' they are is fixed by the simulation chosen.

We start with a broad characterisation of a simulation in terms of two requirements needed to make forward and backward morphisms work as expected. We shall forget, for a while, about the specific case of T^{B}-components, and establish the general setting in terms of arbitrary seeded coalgebras for an extended polynomial Set endofunctor. Afterwards, this will be adapted to T^{B}-components and some refinement examples discussed.

26. SIMULATION. Let T be an extended polynomial Set endofunctor. A *simulation* is a preorder $\leq_{\mathsf{T}} \subseteq \mathsf{T}X \times \mathsf{T}X$, natural in X, such that

- For any $x \in X$ and $x_1, x_2 \in \mathsf{T}X$,

$$x \in_{\mathsf{T}} x_1 \ \wedge \ x_1 \leq_{\mathsf{T}} \ x_2 \ \ \Rightarrow \ \ x \in_{\mathsf{T}} x_2 \tag{7.9}$$

- For any function $h : X \longrightarrow Y$, $\mathsf{T}h$ preserves $\leq$, *i.e.*

$$x_1 \leq_{\mathsf{T}} x_2 \ \ \Rightarrow \ \ (\mathsf{T}h)\, x_1 \leq_{\mathsf{T}} (\mathsf{T}h)\, x_2 \tag{7.10}$$

27. FORWARD AND BACKWARD MORPHISMS. Let T be an extended polynomial functor on Set and consider two seeded T-coalgebras $\alpha : U \longrightarrow \mathsf{T}U$ and $\beta : V \longrightarrow \mathsf{T}V$. A *forward* morphism $h : \alpha \longrightarrow \beta$ with respect to a simulation preorder $\leq$, is a function from U to V such that

$$\mathsf{T}\, h \cdot \alpha \ \leq \ \beta \cdot h$$

where $\leq$ is the pointwise extension of the (equally named) simulation preorder. Dually, h is called a *backwards* morphism if

$$\beta \cdot h \ \leq \ \mathsf{T}\, h \cdot \alpha$$

Let us now show that such morphisms compose and, afterwards, that they can be taken as witnesses of refinement situations.

28. LEMMA. For T an extended polynomial functor in Set, T-coalgebras and forward (respectively, backward) morphisms define a category.

Proof. In both cases, identities are the identities on the carrier and composition is inherited from Set. What remains to be shown is that the composition of forward (respectively, backward) morphisms yields again a forward (respectively, backward) morphism. So, let $h : \alpha \longrightarrow \beta$ and $k : \beta \longrightarrow \gamma$ be two forward morphisms. Then

$$
\begin{array}{ll}
 & \mathsf{T}(k \cdot h) \cdot \alpha \\
= & \qquad \{ \ \mathsf{T} \text{ functor} \ \} \\
 & \mathsf{T}k \cdot (\mathsf{T}h \cdot \alpha) \\
\leq & \qquad \{ \ h \text{ forward and } \leq \text{ simulation (7.10)} \ \} \\
 & \mathsf{T}k \cdot (\beta \cdot h) \\
= & \qquad \{ \ \cdot \text{ associative} \ \} \\
 & (\mathsf{T}k \cdot \beta) \cdot h \\
\leq & \qquad \{ \ k \text{ forward} \ \} \\
 & (\gamma \cdot k) \cdot h \\
= & \qquad \{ \ \cdot \text{ associative} \ \} \\
 & \gamma \cdot (k \cdot h)
\end{array}
$$

The proof is similar for the backward case:

$$
\begin{array}{ll}
 & \gamma \cdot (k \cdot h) \\
= & \qquad \{ \ \cdot \text{ associative} \ \} \\
 & (\gamma \cdot k) \cdot h \\
\leq & \qquad \{ \ k \text{ backward} \ \} \\
 & (\mathsf{T}k \cdot \beta) \cdot h \\
= & \qquad \{ \ \cdot \text{ associative} \ \} \\
 & \mathsf{T}k \cdot (\beta \cdot h) \\
\leq & \qquad \{ \ h \text{ backward and } \leq \text{ simulation (7.10)} \ \} \\
 & \mathsf{T}k \cdot \mathsf{T}h \cdot \alpha \\
= & \qquad \{ \ \mathsf{T} \text{ functor} \ \} \\
 & \mathsf{T}(k \cdot h) \cdot \alpha
\end{array}
$$

□

29. LEMMA. Let T be an extended polynomial functor in Set, and α and β two T-coalgebras as above. Let $\longrightarrow_\alpha$ and $\longrightarrow_\beta$ denote the corresponding transition relations. A forward (respectively, backward) morphism $h : \alpha \longrightarrow \beta$ preserves (respectively, reflects) such transition relations.

Proof. Preservation follows from

$$
\begin{array}{cl}
& u \longrightarrow_\alpha u' \\
\equiv & \quad \{ \longrightarrow \text{ definition} \} \\
& u' \in_{\mathsf{T}} \alpha\, u \\
\Rightarrow & \quad \{ \in_{\mathsf{T}} \text{ property (§24)} \} \\
& h\, u' \in_{\mathsf{T}} (\mathsf{T}h \cdot \alpha)\, u \\
\equiv & \quad \{ h \text{ forward and simulation definition (7.9)} \} \\
& h\, u' \in_{\mathsf{T}} (\beta \cdot h)\, u \\
\equiv & \quad \{ \cdot \text{ associative and } \longrightarrow \text{ definition} \} \\
& h\, u \longrightarrow_\beta h\, u'
\end{array}
$$

To establish reflection suppose that $h\ u \longrightarrow_\beta v'$, *i.e.*, $v' \in_{\mathsf{T}} (\beta \cdot h)\ u$. As h is a backward morphism we have $\beta \cdot h \leq \mathsf{T}\ h \cdot \alpha$, which, together with requirement (7.9) on the definition of a simulation entails, $v' \in_{\mathsf{T}} (\mathsf{T}\ h \cdot \alpha)\ u$. This implies the existence of a state $u' \in U$ such that $v' = h\ u'$ and $u' \in_{\mathsf{T}} \alpha\ u$, *i.e.*, $u \longrightarrow_\alpha u'$.

□

30. A TOO GENEROUS SIMULATION. Let us come back to T^{B}-components. Consider components p and q. For each simulation $\leq$, a forward morphism $h : U_p \longrightarrow U_q$ satisfies

$$\mathsf{B}(h \times \mathsf{id}) \cdot a_p \ \leq\ a_q \cdot (h \times id) \tag{7.11}$$

Consider, now, the following definition of a simulation $\leq_{\mathsf{T}}$:

$$x \leq_{\mathsf{T}} y \quad \text{iff} \quad \forall_{e \in_{\mathsf{T}} x}.\ e \in_{\mathsf{T}} y$$

which captures a sort of T-*structural* inclusion and can be easily proved to be a simulation by induction of the structure of T. If one instantiates $\leq$ in 7.11 by $\leq_{\mathsf{B}(\mathsf{Id} \times O)}$, h will not provide the expected results. Clearly, p transitions from a state u will be a subset of q transitions from $h\ u$, but this will happen regardless of the corresponding outputs! The point is that the transition relation being preserved by such an h is not $\xrightarrow{\langle i,o \rangle}_p$, but simply $\longrightarrow_p$ which does not corresponds to the dynamics of T^{B} coalgebras.

31. COMPONENT SIMULATIONS. The discussion above suggests an additional requirement on simulations for T^{B} components: their definition on a constant functor $\underline{K}$ should be the equality on set K, *i.e.*,

$$x \leq_{\underline{K}} y \quad \mathsf{iff} \quad x =_K y \tag{7.12}$$

Therefore, transitions with different O-labels cannot be related. We can now define refinement in this setting and discuss a few examples.

32. DEFINITION. A T^{B}-component p is a *forward* (respectively, *backward*) refinement of another component q, wrt a simulation preorder R,

$$q \gtrsim_R p \qquad \text{(respectively,} \qquad q \lesssim_R p \text{)}$$

if there exists a forward (respectively, backward) morphism h from p to q.

33. EXAMPLES. A first example of a simulation relation is the following:

$$\begin{array}{rcl}
x \subseteq_{\mathsf{Id}} y & \mathsf{iff} & x = y \\
x \subseteq_{\underline{K}} y & \mathsf{iff} & x =_K y \\
x \subseteq_{\mathsf{T}_1 \times \mathsf{T}_2} y & \mathsf{iff} & \pi_1\, x \subseteq_{\mathsf{T}_1} \pi_1\, y \ \wedge \ \pi_2\, x \subseteq_{\mathsf{T}_2} \pi_2\, y \\
x \subseteq_{\mathsf{T}_1 + \mathsf{T}_2} y & \mathsf{iff} & \begin{cases} x = \iota_1\, x' \wedge y = \iota_1\, y' & \Rightarrow x' \subseteq_{\mathsf{T}_1} y' \\ x = \iota_2\, x' \wedge y = \iota_2\, y' & \Rightarrow x' \subseteq_{\mathsf{T}_2} y' \end{cases} \\
x \subseteq_{\mathsf{T}^{\underline{K}}} y & \mathsf{iff} & \forall_{k \in K} .\ x\, k \subseteq_{\mathsf{T}} y\, k \\
x \subseteq_{\mathcal{P}\mathsf{T}} y & \mathsf{iff} & \forall_{e \in x} \exists_{e' \in y} .\ e \subseteq_{\mathsf{T}} e'
\end{array}$$

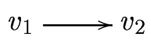
A *forward* refinement of nondeterministic components based on $\subseteq_{\mathsf{T}}$ captures the classical notion of *nondeterminism reduction*. However, this preorder can be tuned to more specific cases. For example, the following 'failure forcing' variant — $\subseteq^E_{\mathsf{T}}$, where E stands for 'emptyset' —- guarantees that the concrete component fails no more than the abstract one. It is defined as $\subseteq_{\mathsf{T}}$ by replacing the clause for the powerset functor by

$$x \subseteq^E_{\mathcal{P}\mathsf{T}} y \quad \mathsf{iff} \quad (x = \emptyset \Rightarrow y = \emptyset) \ \wedge \ \forall_{e \in x} \exists_{e' \in y} .\ e \subseteq_{\mathsf{T}} e'$$

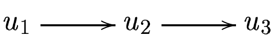
Relation $\subseteq_{\mathsf{T}}$ is inadequate for partial components: both forward or backward refinement collapse into bisimilarity. It may be, however, tuned to capture partiality: relation $\subseteq^F_{\mathsf{T}}$ (F standing for 'failure') replaces the sum clause in $\subseteq_{\mathsf{T}}$ by

$$x \subseteq^F_{\mathsf{T}_1 + \mathsf{T}_2} y \quad \mathsf{iff} \quad \begin{cases} x = \iota_1\, x' \wedge y = \iota_1\, y' & \Rightarrow x' \subseteq_{\mathsf{T}} y' \\ x = \iota_2\, * & \Rightarrow \mathsf{true} \end{cases}$$

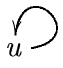
Forward refinement of $\mathsf{T}^{\mathsf{Id}+\mathbf{1}}$-components, with respect to $\subseteq_{\mathsf{T}}^{F}$, entails definition: the 'implementation' is less defined than the 'specification', but they agree on the common transitions. Backward refinement swaps this relation, but only for *strictly* partial components, *i.e.*, components which inevitably fail or deadlock at some point of their evolution. It does the same in the general case but only *up to bisimilarity*, a fact that can probably entail a broader definition of refinement. To see this consider the following diagrams representing the dynamics of four partial components with trivial input and output (*i.e.*, $I = O = \mathbf{1}$),



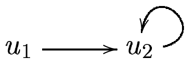
(q)



(p_1)



(p_2)



(p_3)

While there exist backward refinements from either p_1 or p_3 to q, it is not possible to define a backward morphism from p_2 to q (the only possible choice maps state u into v_1). Components p_2 and p_3 are, however, bisimilar (the correspondence $u_1, u_2 \mapsto u$ is a comorphism).

Note that, in general, $q \gtrsim p$ does not entail $p \lesssim q$, even for the same simulation order. On the other hand, it may be easily checked that the anti-symmetric closure of any of the simulation preorders considered in this paragraph entails bisimilarity.

34. APPLICATIONS. The component calculus developed in chapter 5 can be extended with some inequations, capturing refinement situations. Recall, for example, the discussion in §5.15 on the existence of final object in Bh_{B}. For non trivial monads, equation

$$\ulcorner !_I \urcorner \sim p \,;\, \ulcorner !_O \urcorner$$

fails because the right hand side may fail (whenever p does). Function $! : U_p \times \mathbf{1} \longrightarrow \mathbf{1}$ is, however, a forward morphism, with respect to $\subseteq_{\mathsf{T}}^{F}$ for partial components, or to both $\subseteq_{\mathsf{T}}$ and $\subseteq_{\mathsf{T}}^{E}$ for nondeterministic ones. For this last case, note that $\overline{a}_{\ulcorner !_O \urcorner} \cdot ! = \lambda\, i \in I.\ \{*\}$, whereas $\mathsf{B}(! \times \mathsf{id})^I \cdot \overline{a}_{p;\ulcorner !_O \urcorner}\, \langle u, * \rangle$ equals

$$\lambda\, i \in I\ .\ \begin{cases} \{*\} & \text{iff } \overline{a}_p(u)(i) \neq \emptyset \\ \emptyset & \text{iff } \overline{a}_p(u)(i) = \emptyset \end{cases}$$

Therefore,

$$\ulcorner !_I \urcorner \gtrsim p \,;\, \ulcorner !_O \urcorner \tag{7.13}$$

Similarly, cancellation for $\boxtimes$ (see §5.39), is, in general, a refinement result:

$$p \gtrsim \langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner \tag{7.14}$$

Yet another example is given by the (pseudo) naturality of $\ulcorner \triangle \urcorner$ (see §5.40), which could be written as

$$\ulcorner \triangle \urcorner \,;\, (p \boxtimes p) \;\gtrsim\; p \,;\, \ulcorner \triangle \urcorner \tag{7.15}$$

35. DISCUSSION. The simulation preorders introduced above are defined in terms of the structure of extended polynomial functors. Thus, they apply to T^{B}-components where B is either the *finite powerset* or the *maybe* monad. They can be easily extended to the *sequence* monad and, of course, *deterministic* components simply cannot be (further) refined. One may ask, however, how to deal with the *monoidal stamping* monad or, in general, in cases behaviour is captured by monads which are not entirely determined by the structure of the underlying functor. For $\mathsf{B} = \mathsf{Id} \times M$, simulations could be defined if M carries additionally a preorder structure, such that 'costs' of two transitions could be compared. But such simulations need further study.

For the simulation preorders introduced in §33, both $\gtrsim$ and $\lesssim$ can be proved to be *pre-congruences* with respect to the component combinators. The difficulty of proving such a result in a more generic formulation, is certainly a point in favour of moving to an order-enriched setting from the outset.

4. Epilogue

Respostas todo o mundo tem, o que demora é o tempo das perguntas.
José Saramago.

Research on *mathematical models* for programming concepts has been a major theme in Computer Science for the last 30 years, in a fruitful blend of theory and practice. Such models not only permit to focus on the fundamental structures involved, abstracting away from the underlying implementation details, but also provide theories to reason rigorously about complex systems. Therefore, as it happens in other fields of engineering, they provide the basic tools and *calculi* to build reliable designs and have them correctly developed.

Faced with the recent paradigm shift in computing from stand-alone to distributed, open and dynamically changeable environments, computer scientists are under pressure to develop new conceptual models. On-going research on *coalgebras* suggests that, at the *specification* level, the duality between algebraic and coalgebraic structures may provide a bridge between models of *static* and *dynamical* systems. At the *programming* level such a duality, in a canonical initial-final specialisation, captures the intuitive symmetry between *data* and *behaviour*, providing the basis for more uniform and generic approaches to systems' construction.

We would like to regard this thesis as a small contribution in that direction. Of course, further research is needed to assess the practical relevance of this work. For example, is the 'glue' provided by the component calculi the appropriate one? How can *coordination* platforms, which usually rely on a *generative* interaction scheme, be described in our framework? Which classes of components deserve a separate study? How can them be classified? What calculi will emerge?

A lot of work remains to be done.

APPENDIX A

Monads

1. Why Monads? The use of monads to structure the denotational semantics of programming languages was proposed in the 80's, by E. Moggi [Mog89, Mog91]. Later the concept was introduced by Wadler [Wad92, Wad95] in programming practice, entailing a rigorous style of combining purely functional programs that mimic impure (side-)effects. The introduction of monads in program calculation — leading to the study of, *e.g.*, monadic catamorphisms and anamorphisms, — is pursued in [Fok94, MJ95] and [Par98], among others. The key idea is that monads permit to encode in abstract terms several kinds of computational effects, such as exceptions, state updating, nondeterminism or continuations. Such effects are represented by a type constructor (*i.e.*, an endofunctor in a suitable category) B so that computations producing values of type O are regarded as terms of type $\mathsf{B}O$. In this way, *values* and *computations* are explicitly distinguished and programs can be thought of as arrows $I \rightarrow \mathsf{B}O$ representing the computation of values of type O from values of type I, while producing some effect described by B. Or, putting it in a different way, output values arise encapsulated (or embedded) in the effect specified by B. Such arrows will be referred to in the sequel as B-computations or *monadic arrows*. Furthermore, B obeys some equational laws on the interaction of values and computations. Formally the notion of a monad is a category theoretical one, defined as follows:

2. Definition. A *monad* in a category C consists of an endofunctor $\mathsf{B} : \mathsf{C} \longrightarrow \mathsf{C}$ and a pair of natural transformations $\mu : \mathsf{B}^2 \Longrightarrow \mathsf{B}$ and $\eta : \mathsf{Id}_{\mathsf{C}} \Longrightarrow \mathsf{B}$ understood as an associative 'multiplication' and its 'unit', such that the diagrams below commute:

$$\begin{tikzcd}
\mathsf{B}^3 \arrow[r, "\mathsf{B}\mu"] \arrow[d, "\mu\mathsf{B}"'] & \mathsf{B}^2 \arrow[d, "\mu"] \\
\mathsf{B}^2 \arrow[r, "\mu"'] & \mathsf{B}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Id}_{\mathsf{C}}\mathsf{B} \arrow[r, "\eta\mathsf{B}"] \arrow[dr, equal] & \mathsf{B}^2 \arrow[d, "\mu"] & \mathsf{B}\mathsf{Id}_{\mathsf{C}} \arrow[l, "\mathsf{B}\eta"'] \arrow[dl, equal] \\
& \mathsf{B} &
\end{tikzcd}$$

that is,

$$\mu \cdot \eta\mathsf{B} \; = \mu \cdot \mathsf{B}\eta \; = \; \mathsf{id} \qquad \text{(A.1)}$$

$$\mu \cdot \mathsf{B}\mu \; = \mu \cdot \mu\mathsf{B} \tag{A.2}$$

In fact, a monad can be thought of as a monoid in C^{C}, the category of C-endofunctors (recall §2.19). Notice that $\mathsf{B}^2 \; = \; \mathsf{BB}$, and, thus, functor composition, rather than functor product, is used in this definition. Thinking of B as the encapsulation of a computational structure, its unit η represents the minimal such structure when a value $o \in O$ is embedded in $\mathsf{B}O$. Multiplication, on the other hand, flattens computations, providing a way to view a B-effect of a B-effect still as a B-effect.

3. Partiality and Nondeterminism. Two common computational effects are *partiality* and *nondeterminism*. The former models computations that either succeed, returning a value of the given type, or fail raising a simple exception of type $\mathbf{1}$. This is captured by the simplest version of the usual exception monad $\mathsf{B} = \mathsf{Id} + E$, instantiating E with $\mathbf{1}$. In functional programming it is known as the *maybe* monad. *Nondeterminism*, on the other hand, is introduced by the (finite) powerset monad $\mathsf{B} = \mathcal{P}$. Unit and multiplication are defined, in the first case, by $\eta^{-+\mathbf{1}} = \iota_1$ and $\mu^{-+\mathbf{1}} = [\mathsf{id}, \iota_1]$. For the powerset monad, one gets $\eta^{\mathcal{P}-} = \mathsf{sing} = \lambda\, x\, .\; \{x\}$ and $\mu^{\mathcal{P}-} = \bigcup$. A variant to the powerset monad is the *sequence* monad based on $\mathsf{B} = \mathsf{Id}^*$, having as unit the singleton list constructor and, as multiplication, the distributed concatenation of sequences. This entails an 'ordered' view of nondeterminism. A possible interpretation is 'all outputs are possible, but their probability decreases (or increases) along the sequence'.

4. Exponential. The *exponential* monad is based on the functor $\mathsf{B} \; = \; \mathsf{Id}^S$, with unit and multiplication given by

$$\eta_X \;=\; X \xrightarrow{\mathsf{sp}} (X \times S)^S \xrightarrow{{\pi_1}^S} X^S$$

$$\mu_X \;=\; X^{S^S} \xrightarrow{\mathsf{sp}} (X^{S^S} \times S)^S \xrightarrow{\langle \mathsf{ev}, \pi_2 \rangle^S} (X^S \times S)^S \xrightarrow{\mathsf{ev}^S} X^S$$

Going pointwise, the definitions above read:

$$\eta_X\, x \;=\; \lambda s\, .\; x \qquad (\textit{i.e.},\quad \underline{x}\cdot !_S\ )$$

and

$$\mu_X\ m \;=\; \lambda s\, .\; (m\ s)\ s$$

In the context of functional programming this is also known as the *state reader* monad used to encode computations with reading access to a state variable (of type S). When this kind of monadic functions are composed (see §7), the same state is passed as argument to all of them. In fact, a popular use of monads in functional programming is to encapsulate context information, or ‘state’. A more elaborated monad coping with state information is the *state transformer*, based on the functor $\mathsf{B} = (\mathsf{Id} \times S)^S$. In this case $\mu = \mathsf{ev}^S$ and η is simply sp.

5. MONOIDS. In a cartesian category (§2.36), any monoid $\langle M, \theta, e\rangle$ determines a monad based on the functor $\mathsf{B} = M \times \mathsf{Id}$. Unit and multiplication arise as $\eta = \langle \underline{e} \cdot !, \mathsf{id}\rangle$ and $\mu = (\theta \times \mathsf{id}) \cdot \mathsf{a}^\circ$.

6. ADJUNCTIONS. We have seen some common examples of monads in the previous paragraphs. The notion of a monad is, however, extremely general. A basic observation to justify this claim is that any adjunction $\mathsf{T} \dashv \mathsf{S}$ (§2.30) gives rise to a monad based on the composite functor ST. Its unit is simply the unit of the adjunction, and the multiplication is given by $\mathsf{S}\epsilon\mathsf{T}$.

For example, the adjunction between product and exponential reviewed in §2.33, originates the state transformer monad mentioned above, which is described in detail in [Wad92]. The typical case of adjunctions between free and forgetful functors in categories of algebraic structures, gives rise to the *identity* monad. And so on. It is also the case that every monad determines an adjunction.

A related topic is the notion of an *algebra* for a monad. Basically this is defined as an algebra for the monad functor which, additionally, verifies some extra conditions which resemble the definition of *action* for a monoid. We will not pursue this here, but notice that the two categories of algebras that may be associated to a monad B — known as its *Kleisli* and *Eilenberg-Moore* categories — represent, respectively, initial and final solutions to the problem of finding adjunctions that generate B. It should also be referred, to stress the generality of the concept, that any algebra, in the usual sense of the word in Universal Algebra — *i.e.*, every set of operations obeying equational laws — can be regarded as an (algebra for a) monad.

The reader is referred to [Mac71] or [BW85] for standard results on monads and their relevance in algebra. Our concern, in this appendix, is more simply to present the technicalities needed for using monads in the thesis. We will ‘visit’ Kleisli categories, however, as, operationally, they can be regarded as ‘spaces of computations’.

7. KLEISLI COMPOSITION. Monadic arrows, described above, can be composed to build more complex computations from simpler ones. In fact, given $f : I \longrightarrow \mathsf{B}J$ and $g : J \longrightarrow \mathsf{B}O$, their composition $g \bullet f : I \longrightarrow \mathsf{B}O$ is defined by

$$g \bullet f \;=\; I \xrightarrow{f} \mathsf{B}J \xrightarrow{\mathsf{B}g} \mathsf{BB}O \xrightarrow{\mu} \mathsf{B}O$$

This composition law is associative, with η acting as the identity. Therefore, for each monad B, a new category is defined over the objects of C but with B-computations as arrows. This is known as the Kleisli category for B on C.

8. KLEISLI TRIPLES. The composite $\mu \cdot \mathsf{B}\ g$ is sometimes written as $g^* : \mathsf{B}\ J \longrightarrow \mathsf{B}\ O$ and called the *extension* of g. This gives rise to an alternative definition of a monad, known as a *Kleisli triple*. Formally, a *Kleisli triple* consists of an object mapping $\mathsf{B} : \mathrm{obj}(\mathsf{C}) \longrightarrow \mathrm{obj}(\mathsf{C})$, a $\mathrm{obj}(\mathsf{C})$-indexed mapping $\eta_I : I \longrightarrow \mathsf{B}\ I$ natural in I, and an operator $\_^*$ which maps every $f : I \longrightarrow \mathsf{B}\ O$ into $f^* : \mathsf{B}\ I \longrightarrow \mathsf{B}\ O$, satisfying the following axioms:

$$\eta_I{}^* = \mathsf{id}_{\mathsf{B}\ I} \tag{A.3}$$

$$f^* \cdot \eta_I = f \tag{A.4}$$

$$g^* \cdot f^* = (g \cdot f^*)^* \tag{A.5}$$

It is straightforward to show the equivalence between monads and Kleisli triples.

Proof. Given a monad $\langle \mathsf{B}, \eta, \mu \rangle$ a Kleisli triple is obtained by restricting B to objects and defining $f^* = \mu \cdot \mathsf{B}\ f$. For the other direction, suppose given a triple $\langle \mathsf{B}, \eta, \_^* \rangle$. To obtain a monad, first extend B to an endofunctor by defining its action on morphisms as $\mathsf{B}\ (f : I \longrightarrow O) = (\eta_O \cdot f)^*$. Then, define μ_I as $\mathsf{id}^*_{\mathsf{B}\ I}$. The reader can easily verify that, in both cases, the axioms just stated and the ones in §2 are respected.

□

9. STRONG MONADS. In order to handle the presence of ‘context’ and distribute it along computations, monads used in the thesis to capture behavioural models are required to be strong (§3.50). A *strong monad* is simply a monad $\langle \mathsf{B}, \eta, \mu \rangle$ where B is a strong functor (§3.52) and both η and μ strong natural transformations [Koc72]. Therefore, the characterisation law for strong natural transformations (3.20), entails the following additional axioms:

$$\tau_r^{\mathsf{B}} \cdot (\eta \times \mathsf{id}) = \eta \tag{A.6}$$

$$\tau_r^{\mathsf{B}} \cdot (\mu \times \mathsf{id}) = \mu \cdot \mathsf{B}\ \tau_r^{\mathsf{B}} \cdot \tau_r^{\mathsf{B}} \tag{A.7}$$

which express the commutativity of the following diagrams

$$\begin{tikzcd}
\_ \times \_ \arrow[d, "\eta\times\mathsf{id}"'] \arrow[dr, "\eta"] & \\
\mathsf{B} \times \_ \arrow[r, "\tau_r^{\mathsf{B}}"'] & \mathsf{B}(\_ \times \_)
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{B} \times \_ \arrow[rr, "\tau_r^{\mathsf{B}}"] & & \mathsf{B}(\_ \times \_) \\
\mathsf{B}^2 \times \_ \arrow[u, "\mu\times\mathsf{id}"] \arrow[rr, "\tau_r^{\mathsf{B}^2}"] \arrow[dr, "\tau_r^{\mathsf{B}}"'] & & \mathsf{B}^2(\_ \times \_) \arrow[u, "\mu"'] \\
& \mathsf{B}(\mathsf{B} \times \_) \arrow[ur, "\mathsf{B}\ \tau_r^{\mathsf{B}}"'] &
\end{tikzcd}$$

Both laws result from a direct application of (3.20) followed by an unfolding of the strength definition. In particular, recall from §3.52, that τ_r^{Id} is the identity and $\tau_r^{\mathsf{B}^2} = \mathsf{B}\ \tau_r^{\mathsf{B}} \cdot \tau_r^{\mathsf{B}}$.

10. Distributing over products. Applying τ_r and then τ_l, or the other way round, on a product $\mathsf{B}I \times \mathsf{B}J$ yields a result of type $\mathsf{BB}(I \times J)$, which can, then, be flattened, with μ, to $\mathsf{B}(I \times J)$. In most cases, however, the *order* of application is relevant for the outcome.

Formally, we may say that the Kleisli composition of the right with the left strength, gives rise to a natural transformation $\delta_r : \mathsf{B} \times \mathsf{B} \Longrightarrow \mathsf{B}(\mathsf{Id} \times \mathsf{Id})$ whose component on objects I and J is given by

$$\delta r_{I,J} \;=\; \tau r_{I,J} \bullet \tau l_{\mathsf{B}I,J} \tag{A.8}$$

Dually, one may define $\delta l_{I,J} = \tau l_{I,J} \bullet \tau r_{I,\mathsf{B}J}$. Such transformations specify how the monad distributes over the cartesian product and, therefore, represent a sort of sequential composition of B-computations. They relate to strength by

$$\delta r_{I,J} \cdot (\mathsf{id}_{\mathsf{B}I} \times \eta_J) \;=\; \tau r_{I,J} \tag{A.9}$$

$$\delta l_{I,J} \cdot (\eta_I \times \mathsf{id}_{\mathsf{B}J}) \;=\; \tau l_{I,J} \tag{A.10}$$

Whenever δ_r and δ_l coincide, the monad is said to be *commutative* — which is indeed the case of the partiality, powerset and exponential monads (§§3 and 4), but not of the sequence one. It is also the case that a monad generated by a monoid (§5) is commutative if the base monoid is Abelian. In general, however, δ_r and δ_l are related by the following law

$$\delta_r \cdot \mathsf{s} \;=\; \mathsf{B}\,\mathsf{s} \cdot \delta_l \tag{A.11}$$

Proof.

$$
\begin{aligned}
& \delta_r \cdot \mathsf{s} \\
= & \qquad \{ \ \delta_r \text{ definition} \ \} \\
& \mu \cdot \mathsf{B}\ \tau_r \cdot \tau_l \cdot \mathsf{s} \\
= & \qquad \{ \ \tau_r, \tau_l \text{ interaction (§3.52)} \ \} \\
& \mu \cdot \mathsf{B}\ (\mathsf{B}\ \mathsf{s} \cdot \tau_l \cdot \mathsf{s}) \tau_l \cdot \mathsf{s} \\
= & \qquad \{ \ \mathsf{B} \text{ functor} \ \} \\
& \mu \cdot \mathsf{B}\ \mathsf{B}\ \mathsf{s} \cdot \mathsf{B}\ \tau_l \cdot \mathsf{B}\ \mathsf{s} \cdot \tau_l \cdot \mathsf{s} \\
= & \qquad \{ \ \tau_r, \tau_l \text{ interaction (§3.52)} \ \} \\
& \mu \cdot \mathsf{B}\ \mathsf{B}\ \mathsf{s} \cdot \mathsf{B}\ \tau_l \cdot \tau_r \\
= & \qquad \{ \ \mu \text{ natural} \ \} \\
& \mathsf{B}\ \mathsf{s} \cdot \mu \cdot \mathsf{B}\ \tau_l \cdot \tau_r \\
= & \qquad \{ \ \delta_l \text{ definition} \ \} \\
& \mathsf{B}\ \mathsf{s} \cdot \delta_l
\end{aligned}
$$

□

11. COMPOSING MONADS. In what conditions can two monads be composed to yield a new monad? The answer to this question is not trivial because the simple composition of the corresponding functors does not lead, in general, to new monads. The problem is, of course, in the way μ is applied. This entails the need to assert the existence of some sort of distributive law to alter the order of functor application. A general result on this topic can be found in [BW85]. Basically it is proved that given two monads B_1 and B_2 a new monad based on functor $\mathsf{B}_1\ \mathsf{B}_2$ can be defined, provided there exists a natural transformation $\gamma : \mathsf{B}_2\ \mathsf{B}_1 \longrightarrow \mathsf{B}_1\ \mathsf{B}_2$ satisfying a number of conditions. If this is the case, unit and multiplication of the composed monad are given, respectively, by

$$\eta^{\mathsf{B}_1\ \mathsf{B}_2} = \eta^{\mathsf{B}_1} \cdot \eta^{\mathsf{B}_2} \tag{A.12}$$

$$\mu^{\mathsf{B}_1\ \mathsf{B}_2} = \mathsf{B}_1\ \mu^{\mathsf{B}_2} \cdot \mu^{\mathsf{B}_1} \cdot \mathsf{B}_1\ \gamma \tag{A.13}$$

Instead of detailing the general conditions on γ, we shall record here three particular cases which are relevant in the context of this thesis. It should also be mentioned the existence of several other results on monad composition, developed mainly in the context of functional programming (see, for example, [KW92]).

- The *exception* monad (and, of course, the *partiality* monad which is a specialisation of the latter) can be composed with any other monad B, originating $\mathsf{B}\ (\mathsf{Id} + E)$. The distributive law γ is, in this case, defined by

$$\gamma \;=\; \mathsf{B} + E \xrightarrow{[\mathsf{B}\ \iota_1, \eta^{\mathsf{B}} \cdot \iota_2]} \mathsf{B}\ (\mathsf{Id} + E)$$

- The same applies to a monad generated by a monoid M, if B strong. The resulting monad is then $\mathsf{B}\ (M \times \mathsf{Id})$, with

$$\gamma \;=\; \mathsf{B} \times M \xrightarrow{\tau_l^{\mathsf{B}}} \mathsf{B}\ (M \times \mathsf{Id})$$

- Assuming again B strong, it can be composed with the exponential monad, originating B^S, with

$$\gamma \;=\; \mathsf{B}\ (\mathsf{Id}^S) \xrightarrow{\mathsf{sp}} (\mathsf{B}\ (\mathsf{Id}^S) \times S)^S \xrightarrow{(\tau_r^{\mathsf{B}})^S} (\mathsf{B}\ (\mathsf{Id}^S \times S))^S \xrightarrow{(\mathsf{B}\ \mathsf{ev})^S} \mathsf{B}^S$$

Finally, notice that, if B is a commutative monad, so are $\mathsf{B}\ (M \times \mathsf{Id})$, B^S and $\mathsf{B}\ (\mathsf{Id} + \mathbf{1})$. Of course the same is not true for the general exception monad which is itself non commutative.

12. Monad Morphisms. A morphism between two (strong) monads $\langle \mathsf{B}, \eta, \mu \rangle$ and $\langle \mathsf{B}', \eta', \mu' \rangle$, is a strong natural transformation $\psi : \mathsf{B} \Longrightarrow \mathsf{B}'$ such that

$$\psi \cdot \eta \;= \eta' \tag{A.14}$$

$$\psi \cdot \mu \;= \mu' \cdot \psi \cdot \mathsf{B}\ \psi \tag{A.15}$$

These laws, whose diagrams are shown below, are just the 'homomorphism' conditions to guarantee that the monad structure is preserved along the morphism. Notice, in the μ diagram, how a horizontal composition of ψ with itself is decomposed (*cf.* §2.20).

$$\begin{tikzcd}
\mathsf{Id}_{\mathsf{C}} \arrow[r, "\eta^{\mathsf{B}}"] \arrow[d, equal] & \mathsf{B} \arrow[d, "\psi"] \\
\mathsf{Id}_{\mathsf{C}} \arrow[r, "\eta^{\mathsf{B}'}"'] & \mathsf{B}'
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{B} \arrow[d, "\psi"'] & \mathsf{B}^2 \arrow[l, "\mu^{\mathsf{B}}"'] \arrow[r, "\mathsf{B}\psi"] \arrow[d, "\psi;\psi"] & \mathsf{B}\mathsf{B}' \arrow[dl, "\psi"] \\
\mathsf{B}' & \mathsf{B}'^2 \arrow[l, "\mu^{\mathsf{B}'}"] &
\end{tikzcd}$$

By §3.55, being strong also implies that

$$\psi \cdot \tau_r \;= \tau_r^{\mathsf{B}'} \cdot (\psi \times \mathsf{id}) \tag{A.16}$$

An example of a monad morphism from the *sequence* monad to itself is the usual *list reverse* function. As a final remark, note that monad morphisms compose (*cf.*, vertical composition of natural transformations) and there exists, for each monad, a trivial identity morphism. Therefore, monads and their morphisms form a category. Moreover it is immediate to show that the *identity* monad Id is initial in such a category, with $?_{\mathsf{B}} : \mathsf{Id} \longrightarrow \mathsf{B}$ defined as η^{B}, for any other monad B. Similarly, the category has as final object the monad over constant functor $\underline{\mathbf{1}}$.

13. A CHARITABLE ANIMATION. Animating a formal concept in a programming environment may help to build up some intuition about the concept itself. Below we ‘play a little’ with monads as CHARITY objects. As an example, consider the declaration of the *sequence* monad as a ‘data type’

```
data B X = list X.
```

and two functions, encoding its unit and multiplication:

```
def eta: X -> B X
    = x => [x].

def mu: B B X -> B X
    = s => reduce{append, nil} s.
```

where `reduce` implements monoidal reduction as discussed in §E.8. The *maybe* monad, on the other hand, can be defined using the CHARITY primitive type `SF X`, which implements coproduct $X + \mathbf{1}$, together with embeddings `ff` and `ss` from, respectively, $\mathbf{1}$ and X. Thus,

```
data B X = SF X.

def eta: X -> B X
    = x => ss x.

def mu: B B X -> B X
    = ff    => ff | ss x  => x.
```

Right and left strengths for these two monads can be uniformly defined as

```
def taur: B X * C -> B(X * C)
    = (s,c) => B{x => (x,c)} s.

def taul: C * B X -> B(C * X)
    = (c,s) => B{x => (c,x)} s.
```

but the definitions above do not apply, for example, to a monoidal monad. On the other hand, the distributive morphisms, δ_r and δ_l, are uniformly defined for every strong monad:

```
def deltal: B X * B Y -> B(X * Y)
    = (s,t) => mu  B{taul} (taur (s,t)).

def deltar: B X * B Y -> B(X * Y)
    = (s,t) => mu  B{taur} (taul (s,t)).
```

Equally generic is the definition of Kleisli composition:

```
def kleisli{f:X -> B W, g:W -> B Y}: X  -> B Y
    = x => mu  B{g} f x.
```

On top of these constructions, we may ‘ bring life to’ some of the laws introduced in the previous paragraphs. For example, to animate law (A.1) compute both

```
>> mu eta [[1,2],[3],[4,5]].
[[1, 2], [3], [4, 5]] : list(list(int))
```

and

```
>> mu B{eta} [[1,2],[3],[4,5]].
[[1, 2], [3], [4, 5]] : list(list(int))
```

Similarly, a test with laws (A.6) and (A.7), which establish η and μ as strong natural transformations, yields,

```
>> taur (eta 3, 8).
[(3, 8)] : list(int * int)

>> eta (3,8).
[(3, 8)] : list(int * int)
```

and

```
>> taur (mu [[2,3,4], [1,7], [9]], 5).
[(2, 5), (3, 5), (4, 5), (1, 5), (7, 5), (9, 5)] : list(int * int)

>> mu B{taur} taur ([[2,3,4], [1,7], [9]], 5).
[(2, 5), (3, 5), (4, 5), (1, 5), (7, 5), (9, 5)] : list(int * int)
```

Finally, an animation of law (A.9), relating δ_r to τ_r,

```
>> deltar ([1,2,3], eta 5).
[(1, 5), (2, 5), (3, 5)] : list(int * int)

>> taur ([1,2,3], 5).
[(1, 5), (2, 5), (3, 5)] : list(int * int)
```

Observing the application of τ_r, τ_l or $\bullet$ also provides some feedback on these operations. For example,

```
>> taur ([4,6,5],9).
[(4, 9), (6, 9), (5, 9)] : list(int * int)

>> taul (9, [4,6,5]).
[(9, 4), (9, 6), (9, 5)] : list(int * int)


>> kleisli{i => [add_int(i,1), i, sub_int(i,1)], i => [tonat i]} 2.
[succ(succ(succ(zero))), succ(succ(zero)), succ(zero)] : list(nat)
```

and, for the *maybe* monad,

```
>> taur (ss 4, 7).
ss(4, 7) : SF(int * int)

>> taul (7, ss 4).
ss(7, 4) : SF(int * int)

>> taul (7, ff).
ff : SF(int * A)


>> kleisli{i => ss tonat i, n => ss succ n} 3.
ss(succ(succ(succ(succ(zero))))) : SF(nat)

>> kleisli{i => ss tonat i, n => ff} 3.
ff : SF(A)
```

We may also observe δ_l and δ_r coinciding for a commutative monad

```
>> deltal (ss 8, ss 2).
ss(8, 2) : SF(int * int)

>> deltar (ss 8, ss 2).
ss(8, 2) : SF(int * int)

>> deltar (ff, ss 4).
ff : SF(A * int)
```

but otherwise failing:

```
>> deltal([1,2,3],[4,5]).
[(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)] : list(int * int)

>> deltar([1,2,3],[4,5]).
[(1, 4), (2, 4), (3, 4), (1, 5), (2, 5), (3, 5)] : list(int * int)
```

In any case, however, law (A.11) characterises the relationship between the two versions of the distributive law:

```
>> B{swap} deltal ([1,2], [6,1,8]).
[(6, 1), (1, 1), (8, 1), (6, 2), (1, 2), (8, 2)] : list(int * int)

>> deltar swap ([1,2], [6,1,8]).
[(6, 1), (1, 1), (8, 1), (6, 2), (1, 2), (8, 2)] : list(int * int)
```

where `swap` is the commutativity morphism:

```
def swap: X * Y -> Y * X
    = (x,y) => (y,x).
```

APPENDIX B

Bicategories

1. Definition

1. MOTIVATION. Elements of a set are either equal or different. In a category, however, two objects can be different but still essentially the same, in a very precise way. Technically, they are said to be *isomorphic* and the construction witnessing this fact can be made explicit and used in calculations. For example, in Set, $A \times B$ and $B \times A$ are made isomorphic by the (explicitly introduced) bijection $\mathsf{s} = \langle \pi_2, \pi_1 \rangle$. Moreover, objects can be essentially the same in more than one way, as such 'sameness' may be witnessed by different isomorphisms.

In order to extend this view to the level of morphisms, the category has to be equipped with further structure. In particular, this requires arrows between arrows and a corresponding notion of composition. To avoid name clashing, such arrows are usually called 2-cells (objects and object morphisms being similarly named 0-cells and 1-cells, respectively).

In this context, the space of morphisms between any given pair of objects, usually referred to as a (hom-)set, acquires itself the structure of a category. Therefore the basic (1-cell) composition and unit laws become functorial, since they transform both objects (1-cells) and arrows (2-cells) of each (hom-)category in an uniform way. In consequence, 2-cells are composable by two different, but intrinsically related (see §4 below) ways.

Such a structure has the ability to refine the notion of 'sameness' by distinguishing isomorphisms further than the equality level. A further step, in the same spirit, consists of weakening the degree of strictness up to which the usual associative and unit laws for 1-cell composition are supposed to hold. If, in particular, equality is relaxed to isomorphism one ends up with a *bicategory*. Of course, the witnessing isomorphisms have to be explicitly included in the definition as particular, invertible 2-cells. However, there is a price to be paid for this increased expressiveness: such isomorphisms should themselves obey some (coherence) laws to be used in calculations as if they were proper equalities.

The following paragraphs introduce some basic definitions and a few examples of bicategories. This should be enough to understand their use in the thesis, but should not be regarded as a thorough introduction to the topic. The basic reference on bicategories is Bénabou's original paper [Ben67]. Comprehensive accounts can be found in, *e.g.*, [Bor94b] and [Str96].

2. DEFINITION. The underlying structure of a *bicategory* B consists of

- a class of objects A, B, C, ...
- for each pair $\langle A, B\rangle$ of objects, a small (hom-)category $\mathsf{B}(A,B)$ with arrows $p, q, r, ...$ from A to B as objects and arrows $h, k, l, ...$ between them denoted as in, *e.g.*, $h : p \Longrightarrow q$, and referred to as 2-cells. Composition in $\mathsf{B}(A,B)$ is denoted by $\cdot$ and the identity on p, for each $p : A \longrightarrow B$, by $1_p : p \Rightarrow p$.
- for each triple $\langle A, B, C\rangle$ of objects, a composition law given by a (bi)functor

$$;_{A,B,C} : \mathsf{B}(A,B) \times \mathsf{B}(B,C) \longrightarrow \mathsf{B}(A,C)$$

- for each object A, an identity functor

$$I_A : 1 \longrightarrow \mathsf{B}(A,A)$$

  where 1 stands for the final object in the category Cat of small categories.

3. REMARK. From the definition above, 2-cells in B come equipped with two forms of composition, called, respectively, *vertical* and *horizontal* after the respective diagrammatic presentation. Vertical composition is given by the composition law in each hom category. On the other hand, horizontal composition corresponds to the action of functor ; on 2-cells. The action of I_A on the unique object of 1 is the 1-cell $I_A : A \longrightarrow A$, the identity on object A (wrt ; on 1-cells), whereas its action on the unique arrow of 1 is the 2-cell $1_{I_A} : I_A \Rightarrow I_A$, the identity on the 1-cell $I_A : A \longrightarrow A$ (wrt ; on 2-cells, *i.e.*, horizontal 2-cell composition). This is sketched as follows:

$$\begin{tikzcd}
A \arrow[r, bend left, "I_A"{name=U}] \arrow[r, bend right, "I_A"'{name=D}] & A
\arrow[Rightarrow, from=U, to=D, "1_{I_A}"]
\end{tikzcd}$$

4. INTERCHANGE LAW. The two forms of composition mentioned above are related by the equality

$$(k\,;k') \cdot (h\,;h') \;=\; (k \cdot h)\,;(k' \cdot h')$$

which gives an unambiguous meaning to the diagram

$$\begin{tikzcd}
A \arrow[r, bend left=60, "p", ""{name=P, below}] \arrow[r, "q"', ""{name=Q, above}, ""{name=QQ, below}] \arrow[r, bend right=60, "r"', ""{name=R, above}] & B \arrow[r, bend left=60, "p'", ""{name=P2, below}] \arrow[r, "q'"', ""{name=Q2, above}, ""{name=QQ2, below}] \arrow[r, bend right=60, "r'"', ""{name=R2, above}] & C
\arrow[Rightarrow, from=P, to=Q, "h"]
\arrow[Rightarrow, from=QQ, to=R, "k"]
\arrow[Rightarrow, from=P2, to=Q2, "h'"]
\arrow[Rightarrow, from=QQ2, to=R2, "k'"]
\end{tikzcd}$$

The equation arises simply because $;_{A,B,C}$, for each triple $\langle A, B, C\rangle$, is a bifunctor. This is widely used in calculations involving natural transformations (§2.17).

5. Associativity and Unit Laws. The motivation in §1 has hopefully shed some light on why associativity and unit axioms for **;** are supposed to hold only up to isomorphism. Therefore, such axioms are introduced explicitly in the definition of a bicategory as natural isomorphisms. Moreover, they are subject to some suitable coherence laws integrating the definition as well.

6. Definition. A *bicategory* B is defined by the structure introduced in §2 plus the following natural isomorphisms, for each A, B, C and D[1]:

$$\begin{aligned}
\mathsf{a}_{A,B,C,D} &: \ ;_{A,B,D} \circ (\mathsf{Id} \times ;_{B,C,D}) \Longrightarrow \ ;_{A,C,D} \circ (;_{A,B,C} \times \mathsf{Id}) \\
\mathsf{r}_{A,B} &: \ ;_{A,A,B} \circ (I_A \times \mathsf{Id}) \Longrightarrow \mathsf{Id} \\
\mathsf{l}_{A,B} &: \ ;_{A,B,B} \circ (\mathsf{Id} \times I_B) \Longrightarrow \mathsf{Id}
\end{aligned}$$

diagrammatically,

$$\begin{tikzcd}
\mathsf{B}(A,B) \times \mathsf{B}(B,C) \times \mathsf{B}(C,D) \arrow[r, ";_{A,B,C} \times \mathsf{Id}"] \arrow[d, "\mathsf{Id} \times ;_{B,C,D}"'] & \mathsf{B}(A,C) \times \mathsf{B}(C,D) \arrow[d, ";_{A,C,D}"] \\
\mathsf{B}(A,B) \times \mathsf{B}(B,D) \arrow[r, ";_{A,B,D}"'] \arrow[ur, Rightarrow, shorten=20pt, "\mathsf{a}_{A,B,C,D}" description] & \mathsf{B}(A,D)
\end{tikzcd}$$

and

$$\begin{tikzcd}
1 \times \mathsf{B}(A,B) \arrow[d, "I_A \times \mathsf{Id}"'] & \mathsf{B}(A,B) \arrow[l, "\cong"'] \arrow[r, "\cong"] \arrow[d, equal] & \mathsf{B}(A,B) \times 1 \arrow[d, "\mathsf{Id} \times I_B"] \\
\mathsf{B}(A,A) \times \mathsf{B}(A,B) \arrow[r, ";_{A,A,B}"'] \arrow[ur, Rightarrow, shorten=15pt, "\mathsf{r}_{A,B}" description] & \mathsf{B}(A,B) & \mathsf{B}(A,B) \times \mathsf{B}(B,B) \arrow[l, ";_{A,B,B}"] \arrow[ul, Rightarrow, shorten=15pt, "\mathsf{l}_{A,B}" description]
\end{tikzcd}$$

[1] Recall from §2.15, that, in the context of this thesis, $\circ$ (often abbreviated by juxtaposition) denotes functor composition and Id is the identity functor on any category.

The components of these isomorphisms, for $p : A \longrightarrow B, q : B \longrightarrow C, r : C \longrightarrow D$ and $t : D \longrightarrow E$, are, as expected, the invertible 2-cells

$$\begin{aligned} \mathsf{a}_{p,q,r} &: p \,;(q\,;r) \overset{\cong}{\Longrightarrow} (p\,;q)\,;r \\ \mathsf{r}_p &: I_A\,;p \overset{\cong}{\Longrightarrow} p \\ \mathsf{l}_p &: p\,;I_B \overset{\cong}{\Longrightarrow} p \end{aligned}$$

subject to the coherence laws expressed by the commutativity of the following diagrams:

$$\begin{tikzcd}
p\,;(q\,;(r\,;t)) \arrow[rr, "1_p;\mathsf{a}"] \arrow[d, "\mathsf{a}"'] & & p\,;((q\,;r)\,;t) \arrow[d, "\mathsf{a}"] \\
(p\,;q)\,;(r\,;t) \arrow[dr, "\mathsf{a}"'] & & (p\,;(q\,;r))\,;t \arrow[dl, "\mathsf{a};1_t"] \\
& ((p\,;q)\,;r)\,;t &
\end{tikzcd}$$

and

$$\begin{tikzcd}
p\,;(I_B\,;q) \arrow[rr, "\mathsf{a}"] \arrow[dr, "1_p;\mathsf{r}"'] & & (p\,;I_B)\,;q \arrow[dl, "\mathsf{l};1_q"] \\
& p\,;q &
\end{tikzcd}$$

7. 2-CATEGORIES. The structure arising by taking the families of natural isomorphisms a, l and r as mere identities, is called a *2-category*. In this stricter setting the coherence axioms hold automatically.

2. Examples

8. EXAMPLES. A typical example of a 2-category is Cat, with small categories, functors and natural transformations as 0, 1 and 2-cells respectively.

Another trivial example of a bicategory arises by duality. The dual B^{op} of a bicategory B is still a bicategory, formed by reversing the 1-cells. The 2-cells of B, however, remain unchanged (just think on $\mathsf{Cat}^{\mathsf{op}}$).

Finally, any category C can be seen as a special case of a 2-category, and hence of a bicategory, by regarding each homset $\mathsf{C}(A, B)$ as a discrete category.

9. SPANS. A more interesting example of a bicategorical structure is given by the spans of a category C with pullbacks. The construction is as follows: take the objects of C as 0-cells and define 1-cell (f, g) from A to B as a span $\langle X, f, g\rangle$, *i.e.*, a pair of C-arrows, $f : X \longrightarrow A$ and $g : X \longrightarrow B$, with a common domain. Spans compose by pullbacking, *i.e.*, $(f, g)\ ;\ (f', g') = (f \cdot x, g' \cdot y)$, where (x, y) is an arbitrarily specified pullback of (g, f') as in the following diagram:

$$\begin{tikzcd}
 & & P \arrow[dl, "x"'] \arrow[dr, "y"] & & \\
 & X \arrow[dl, "f"'] \arrow[dr, "g"] & & X' \arrow[dl, "f'"'] \arrow[dr, "g'"] & \\
A & & B & & C
\end{tikzcd}$$

The identity on object A is, of course, $(\mathsf{id}_A, \mathsf{id}_A)$, where id_A is the identity on A in C. Now define a 2-cell as a morphism between spans on the same objects, *i.e.*, a C-arrow $k : X \longrightarrow Y$ making the following diagram to commute.

$$\begin{tikzcd}
 & X \arrow[dl, "f"'] \arrow[dr, "g"] \arrow[dd, Rightarrow, "k"'] & \\
A & & B \\
 & Y \arrow[ul, "f'"] \arrow[ur, "g'"'] &
\end{tikzcd}$$

Vertical composition of such morphisms is simply inherited from C. On the other hand, horizontal composition is less obvious. Given $k : (f, g) \Rightarrow (f', g')$ and $l : (m, n) \Rightarrow (m', n')$, as below, their composite $k\ ;\ l$ is the 2-cell $h : (f \cdot x, n \cdot y) \Rightarrow (f' \cdot x', n' \cdot y')$ defined as the unique factorisation through the pullback $\langle P', x', y'\rangle$ of

$k \cdot x$ and $l \cdot y$, as illustrated in the following diagram.

$$\begin{tikzcd}
 & & P \arrow[dl, "x"'] \arrow[dr, "y"] \arrow[dddd, Rightarrow, "h"] & & \\
 & X \arrow[dl, "f"'] \arrow[dr, "g"] \arrow[dd, Rightarrow, "k"'] & & Y \arrow[dl, "m"'] \arrow[dr, "n"] \arrow[dd, Rightarrow, "l"] & \\
A & & B & & C \\
 & X' \arrow[ul, "f'"] \arrow[ur, "g'"'] & & Y' \arrow[ul, "m'"] \arrow[ur, "n'"'] & \\
 & & P' \arrow[ul, "x'"] \arrow[ur, "y'"'] & &
\end{tikzcd}$$

Note that

$$\begin{array}{cl}
 & g' \cdot k \cdot x \\
= & \qquad \{\ k \text{ is a 2-cell } \} \\
 & g \cdot x \\
= & \qquad \{\ P \text{ is a pullback } \} \\
 & m \cdot y \\
= & \qquad \{\ l \text{ is a 2-cell } \} \\
 & m' \cdot l \cdot y
\end{array}$$

and h is actually a 2-cell:

$$\begin{array}{cl}
 & f' \cdot x' \cdot h \\
= & \qquad \{\ h \text{ factorizes } k \cdot x\ \} \\
 & f' \cdot k \cdot x \\
= & \qquad \{\ k \text{ is a 2-cell } \} \\
 & f \cdot x
\end{array}$$

and similarly one can prove $n' \cdot y' \cdot h \ = \ n \cdot y$. Spans are a useful device to generalise relations to an arbitrary category — recall, *e.g.*, the general formulation of *bisimulation* in §3.23. As pullbacks are defined up to isomorphism and in the definition of **;** they are arbitrarily chosen, the corresponding associativity also holds only up to isomorphism. Having chosen the A-identity span as the identity on A, the isomorphisms corresponding to the left or right unit laws, become simply the identity natural transformations.

10. Partial Maps. Another typical use of spans arises in the definition of sets and partial maps. In fact a partial map from a set A to a set B may be regarded as an isomorphism class of spans on A and B, where the first component (dom) is a monomorphism, *i.e.*,

$$\begin{tikzcd}
& U \arrow[dl, hook', "\mathrm{dom}"'] \arrow[dr, "f"] & \\
A & & B
\end{tikzcd}$$

Let $[\langle U, \mathrm{dom}, f\rangle]_{\cong}$ stand for the isomorphism class of $\langle U, \mathrm{dom}, f\rangle$. A morphism between two partial maps $[\langle U, \mathrm{dom}, f\rangle]_{\cong}$ and $[\langle V, \mathrm{dom}', f'\rangle]_{\cong}$ from A to B is just a 2-cell between the corresponding spans, *i.e.*,

$$\begin{tikzcd}
& U \arrow[dl, hook', "\mathrm{dom}"'] \arrow[dr, "f"] \arrow[dd, Rightarrow, "p"'] & \\
A & & B \\
& V \arrow[ul, hook, "\mathrm{dom}'"] \arrow[ur, "f'"'] &
\end{tikzcd}$$

Wherever it exists, this arrow is unique, which makes Par, the category of partial maps, a locally-ordered bicategory (see [Car87] for details).

3. Further Structure

11. Stepping Down. Cell-reindexing is a simple way to *step down* from a bicategory to a category. One forms the new category by taking the original 1-cells as objects and the 2-cells as arrows. This new category still carries the 'genetic inheritance' of the original one, in the form of some additional structure. Here is a particularly simple, but illustrative example.

Think, first, in an ordinary (1-)category with an unique object A. One may form a new category taking as objects the (auto)morphisms on A and as arrows the morphisms between them. As we have begun with an ordinary (1-)category the latter simply do not exist and, therefore, the result is just the *set* (*i.e.*, the discrete or 0-category) of automorphisms. Their composition appears now as a binary associative operation, with identity, on the set, which, thanks to this “inheritance”, becomes a *monoid*.

If the same procedure is applied to a bicategory also with just one object, the original 2-cells become the arrows of an ordinary category, and composition amounts to the original vertical composition. Notice that, again, the obtained category inherits some additional structure: since its objects are arrows of the original bicategory, it gets for free a ‘multiplication’ at the object level. The result is, of course, a *monoidal* category.

12. HOMOMORPHISMS OF BICATEGORIES. Just as bicategories generalise categories, a bicategory homomorphism arises as a generalisation of the notion of a functor. The cornerstone of such an extension is compatibility with the 2-cell structure.

Again the functoriality axioms can be required to hold at different levels of strictness. Therefore, they are built into the definition as particular natural transformations obeying some coherence laws.

In the definition below nothing but naturality is assumed about such transformations. The kind of homomorphism defined is called a *lax functor* and is the standard notion of bicategory homomorphism, consistent with the weak approach to n-categories (see §17 further on).

Requiring $\mathsf{m}_{A,B,C}$ and u_A below to be isomorphisms amounts to the definition of a *pseudo-functor*, *i.e.*, a functor being functorial up to isomorphism. Hence, $\mathsf{T}f \fatsemi \mathsf{T}g \;\cong\; \mathsf{T}(f \,;\, g)$ and $\mathsf{T}I \;\cong\; \mathsf{T}I'$. Pseudo-functors of the form $\mathsf{I} : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathsf{Cat}$ are well-known in applications of category theory to computer science to model *indexing* situations. In such a context they are named *indexed categories* and shown to be equivalent to *fibrations* [Gro70] a more convenient tool to achieve the same (see the recent book by B. Jacobs [Jac99a] for systematic applications to logic and type theory or [Str99] for a tutorial).

Finally, a stricter approach will enforce $\mathsf{m}_{A,B,C}$ and u_A as effective identities, making all the axioms to hold on the nose. This defines a *2-functor*, which is the usual notion of a homomorphism of 2-categories.

13. DEFINITION. Let B and B' be two bicategories. A *homomorphism* T from B to B', is called a *lax functor* and consists of

- a function T mapping objects of C into objects of C',
- for each pair $\langle A, B\rangle$ of objects, a functor $\mathsf{T}_{A,B} : \mathsf{B}(A,B) \longrightarrow \mathsf{B}'(\mathsf{T}A, \mathsf{T}B)$
- for each triple $\langle A, B, C\rangle$ of objects, a natural transformation $\mathsf{m} : \fatsemi \circ (\mathsf{T}_{A,B} \times \mathsf{T}_{B,C}) \Longrightarrow \mathsf{T}_{A,C} \circ \,;$ whose components are 2-cells $\mathsf{m}_{p,q} : \mathsf{T}p \,;'\, \mathsf{T}q \longrightarrow$

$\mathsf{T}(p\,;q)$, for each $p : A \longrightarrow B$ and $q : B \longrightarrow C$. In a diagram:

$$\begin{tikzcd}
\mathsf{B}(A,B) \times \mathsf{B}(B,C) \arrow[r, ";_{A,B,C}"] \arrow[d, "\mathsf{T}_{A,B} \times \mathsf{T}_{B,C}"'] & \mathsf{B}(A,C) \arrow[d, "\mathsf{T}_{A,C}"] \\
\mathsf{B}'(\mathsf{T}A,\mathsf{T}B) \times \mathsf{B}'(\mathsf{T}B,\mathsf{T}C) \arrow[r, ";'_{\mathsf{T}A,\mathsf{T}B,\mathsf{T}C}"'] \arrow[ur, Rightarrow, "\mathsf{m}_{A,B,C}"] & \mathsf{B}'(\mathsf{T}A,\mathsf{T}C)
\end{tikzcd}$$

- for each object A, a natural transformation $\mathsf{u} : I'_{\mathsf{T}A} \Longrightarrow \mathsf{T}_{A,A} \circ I_A$ as in diagram

$$\begin{tikzcd}
1 \arrow[r, "I_A"] \arrow[d, equal] & \mathsf{B}(A,A) \arrow[d, "\mathsf{T}_{A,A}"] \\
1 \arrow[r, "I'_{\mathsf{T}A}"'] \arrow[ur, Rightarrow, "\mathsf{u}"] & \mathsf{B}'(\mathsf{T}A,\mathsf{T}A)
\end{tikzcd}$$

subject to the coherence laws expressed by the commutativity of the following diagrams:

$$\begin{tikzcd}
\mathsf{T}p\,;'(\mathsf{T}q\,;'\mathsf{T}r) \arrow[r, "\mathsf{Id};'\mathsf{m}"] \arrow[d, "\mathsf{a}'"'] & \mathsf{T}p\,;'\mathsf{T}(q\,;r) \arrow[r, "\mathsf{m}"] & \mathsf{T}(p\,;(q\,;r)) \arrow[d, "\mathsf{Ta}"] \\
(\mathsf{T}p\,;'\mathsf{T}q)\,;'\mathsf{T}r \arrow[r, "\mathsf{m};'\mathsf{Id}"'] & \mathsf{T}(p\,;q)\,;'\mathsf{T}r \arrow[r, "\mathsf{m}"'] & \mathsf{T}((p\,;q)\,;r)
\end{tikzcd}$$

and

$$\begin{tikzcd}
I'_{\mathsf{T}A}\,;'\mathsf{T}p \arrow[r, "\mathsf{r}'"] \arrow[d, "\mathsf{u};'\mathsf{Id}"'] & \mathsf{T}p \arrow[dd, equal] & \mathsf{T}p\,;'I'_{\mathsf{T}B} \arrow[l, "\mathsf{l}'"'] \arrow[d, "\mathsf{Id};'\mathsf{u}"] \\
\mathsf{T}I_A\,;'\mathsf{T}p \arrow[d, "\mathsf{m}"'] & & \mathsf{T}p\,;'\mathsf{T}I_B \arrow[d, "\mathsf{m}"] \\
\mathsf{T}(I_A\,;p) \arrow[r, "\mathsf{Tr}"'] & \mathsf{T}p & \mathsf{T}(p\,;I_B) \arrow[l, "\mathsf{Tl}"]
\end{tikzcd}$$

14. AN CHEAP EXAMPLE. Take the singleton set $\mathbf{1}$ as a discrete (bi)category and let B be any bicategory. Then any lax-functor $\mathsf{M} : \mathbf{1} \longrightarrow \mathsf{B}$ is nothing more than a *monad* in B. The good old friend of functional programmers is around the corner, just by instantiating B with Cat.

15. Further Structure. A similar construction yields corresponding notions of natural transformation in a bicategorical setting. Components of a natural transformation τ between, say, (lax-)functors T and R are, of course, 1-cells indexed by the objects. However, the naturality requirement is, again, introduced as the following family of natural transformations $\mathsf{n}_{A,B}$

$$\begin{tikzcd}
\mathsf{B}(A,B) \arrow[r, "\mathsf{T}_{A,B}"] \arrow[d, "\mathsf{R}_{A,B}"'] & \mathsf{B}'(\mathsf{T}A,\mathsf{T}B) \arrow[d, "\mathsf{B}'(\mathsf{T}A{,}\tau_B)"] \\
\mathsf{B}'(\mathsf{R}A,\mathsf{R}B) \arrow[r, "\mathsf{B}'(\tau_A{,}\mathsf{R}B)"'] \arrow[ur, Rightarrow, "\mathsf{n}_{A,B}"] & \mathsf{B}'(\mathsf{T}A,\mathsf{R}B)
\end{tikzcd}$$

where $\mathsf{B}(X,p) : \mathsf{B}(X,A) \longrightarrow \mathsf{B}(X,B)$ is the functor induced by the 1-cell $p : A \longrightarrow B$, for a given X, and, contravariantly, $\mathsf{B}(p,X) : \mathsf{B}(B,X) \longrightarrow \mathsf{B}(A,X)$.

One can go further and define a morphism between this kind of transformations

$$\begin{tikzcd}
\mathsf{T} \arrow[r, bend left, "\tau"] \arrow[r, bend right, "\tau'"'] \arrow[r, phantom, "\Downarrow \sigma"] & \mathsf{R}
\end{tikzcd}$$

whose components are 2-cells connecting, for each object A, τ_A to τ'_A. Such morphisms are known as *modifications* and, in particular, allow for the definition of the analogue of a functor category between two bicategories. Given B and B', a *functor bicategory* arises taking (lax-)functors, transformations and modifications as 0, 1 and 2-cells, respectively.

Having built all this structure, one can define what adjunctions are and, consequently, what limits mean, in it. There is also a notion of *representable* and an analogue of Yoneda lemma, which is used in the proof of the *coherence theorem* [Str96] asserting the possibility of reducing any bicategory to a (bi)equivalent 2-category. Although this will not be pursued here, we should remark that suitable generalisations of familiar categorical constructions emerge as expected, respecting the 2-cell structure and eventually relaxing the conditions up to which axioms are verified. Coherence requirements, however, may become rather heavy to state and prove.

Finally, notice that the fact that Cat is a (particular case of a) bicategory allows one to borrow common categorical constructions and have them interpreted in an arbitrary bicategory. For example, a pair of 1-cells $p : A \longrightarrow B$ and $q : B \longrightarrow A$ equipped with a 2-cell isomorphism $i : I_A \Rightarrow p \,;\, q$ in the hom-category $\mathsf{B}(A,A)$, and another one $j : I_B \Rightarrow q \,;\, p$ in the hom-category $\mathsf{B}(B,B)$, define an *equivalence* between objects A and B. In fact an equivalence of categories is just an instantiation of this notion in Cat.

16. COHERENCE. A final remark on *coherence* is in order. Coherence laws arise in the definition of a bicategory as well as in other related structures. A particularly familiar example in computer science is the case of monoidal categories, which, as mentioned above, can be thought of as born out of bicategories.

In a sense, coherence axioms are a price to be paid for the increased expressive power originated by weakening the defining structural properties. Think, for example, of the coherence diagram for associativity in definition §6. It identifies the basic ways in which the composition of 4 arrows can be parenthesised and relates them through a. To ensure that all such ways are unique is precisely the reason to enforce the commutativity of the diagram. By such a coherence axiom one knows that, in a bicategory, any two natural isomorphisms built out of a, r and l, by the composition and unit operations, actually coincide. That is to say, weakening has caused no special (calculational) damage.

The difficult question is: why is this so? Of course there are standard results asserting the fact (*e.g.*, Mac Lane's coherence theorem for monoidal categories [Mac71] or Bénabou's result for bicategories [Ben67]) but it would help to have a deeper understanding of the origins of coherence axioms.

Something that may help to build up intuition, is the observation, due to J. Dolan and J. Baez, among others, that an operation automatically satisfies all the (relevant) coherence laws if defined by an universal property. For example, Set has all finite products which are defined by an universal property (§2.22). Moreover they are unique up to isomorphism. If one takes the product $A \times B$ for every pair of sets, making cartesian product an operation, three natural isomorphisms, expressing associativity, left and right units, get defined canonnically. Such isomorphisms verify the coherence axioms for a tensor product turning, therefore, $\langle \mathsf{Set}, \times \rangle$ into a monoidal category. One may therefore conclude that the usual definition of a monoidal category, with the explicit coherence axioms, amounts to the fact that any monoidal structure defined by universal properties automatically satisfies such axioms.

17. n-CATEGORIES. The whole subject of bicategories is much wider than we have been able to glimpse in this appendix. In fact, both generalisations embodied in the notion of a bicategory (*i.e.*, the introduction of arrows between arrows and the weakening of the degree of strictness up to which axioms hold) can be pursued further. Recall that the justification for the introduction of 2-cells is the possibility of having arrows around that are isomorphic rather than merely equal. Once all arrows become objects, the sentence will still make sense, as a justification for the introduction of categories themselves in the first place. One can easily imagine this process going on: considering 3-cells as 2-cell morphisms and so on. On the other hand, the relevant equations may be taken to hold up to isomorphism in the immediately lower level or, even more generally, up to an arbitrary arrow.

This is the broader context of *n-categories* [Bae97], whose basic claim is that equations should hold *on the nose* (*i.e.*, up to equality) only at the top level, *i.e.*, between n-cells. Therefore laws concerning k-cells, for $k < n$, should always be expressed as $(k+1)$-equivalences. In this context an equivalence between $(n-1)$-cells is an invertible n-cell whereas an equivalence between k-cells, for $k < n$, is just a $(k+1)$-cell invertible up to equivalence.

The framework is very expressive — in practice often things are only true up to (a suitable notion of) isomorphism, and sometimes only up to other things. But some care is needed to avoid getting puzzled by coherence conditions. The expressive power of *n-categories* is well illustrated by noting that a $(n+1)$-category with only one object can always be regarded as a special kind of a n-category. This was exactly what we have seen, for $n = 1$ and $n = 2$, in §11.

APPENDIX C

Context Laws

1. Preliminaries

1. MOTIVATION. The approach to components modelling presented in this thesis, namely in chapters 5 and 6, is parametric on a behaviour functor B which is required to form a, usually commutative, strong monad. It is therefore not surprising that the proofs of most properties of component's combinators involve common 'housekeeping' morphisms such as associativity (a), commutativity (s) and exchange (xr, xl and m), interplaying with monad unit, multiplication and strength.

This appendix collects and proves a number of generic laws we have found to be useful in trying to establish the envisaged component's calculi. Since we have proceeded on a 'demand driven' basis we cannot claim that such a repertoire of laws is complete or exhaustive. Despite this fact, we believe these laws may be of use in other situations of context manipulation in expressions involving strength and the monad definitional morphisms.

2. STARTING POINT. Our starting point consists of the strength and monad axioms reviewed, respectively, in §3.52 and appendix A. Let us recall them briefly here for easier reference. Let B be a strong functor. Then the following laws hold:

- τ_r and τ_l unit and associativity (equations (3.14), (3.15) and (3.16), (3.17), respectively):

$$\mathsf{Bl} \cdot \tau_r = \mathsf{l} \tag{C.1}$$

$$\mathsf{Br} \cdot \tau_l = \mathsf{r} \tag{C.2}$$

$$\mathsf{Ba}^\circ \cdot \tau_r = \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \tag{C.3}$$

$$\mathsf{Ba} \cdot \tau_l = \tau_l \cdot (\mathsf{id} \times \tau_l) \cdot \mathsf{a} \tag{C.4}$$

- τ_r and τ_l are natural:

$$\tau_r \cdot (\mathsf{B}f \times g) = \mathsf{B}(f \times g) \cdot \tau_r \tag{C.5}$$

$$\tau_l \cdot (f \times \mathsf{B}g) = \mathsf{B}(f \times g) \cdot \tau_l \tag{C.6}$$

- τ_r and τ_l commute with each other:

$$\mathsf{Bs} \cdot \tau_r \;= \tau_l \cdot \mathsf{s} \tag{C.7}$$

As usual, different, but equivalent, formulations of these laws arise by pre- and post-composition of both sides with suitable isomorphisms. For example, post-composition of both sides of (C.3) with Ba and simultaneous pre-composition with a leads to

$$\tau_r \cdot \mathsf{a} \;= \mathsf{Ba} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \tag{C.8}$$

Similarly, from (C.4), one gets

$$\tau_l \cdot \mathsf{a}^\circ \;= \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times \tau_l) \tag{C.9}$$

Since s is its own inverse, the definitions of τ_r in terms of τ_l (and vice-versa) are a direct consequence of (C.7):

$$\tau_r \;= \mathsf{Bs} \cdot \tau_l \cdot \mathsf{s} \tag{C.10}$$

$$\tau_l \;= \mathsf{Bs} \cdot \tau_r \cdot \mathsf{s} \tag{C.11}$$

Other useful results are derived from the above. For example, the unit law entails

$$\mathsf{B}\pi_1 \cdot \tau_r \;= \pi_1 \tag{C.12}$$

$$\mathsf{B}\pi_2 \cdot \tau_l \;= \pi_2 \tag{C.13}$$

Proof.

$$\begin{aligned}
\mathsf{B}\pi_1 \cdot \tau_r &= \mathsf{B}(\mathsf{l} \cdot (\mathsf{id}\times !)) \cdot \tau_r && \{\ \pi_1 \text{ definition }\} \\
&= \mathsf{Bl} \cdot \mathsf{B}(\mathsf{id}\times !) \cdot \tau_r && \{\ \mathsf{B} \text{ functor }\} \\
&= \mathsf{Bl} \cdot \tau_r \cdot (\mathsf{id}\times !) && \{\ \tau_r \text{ natural (C.5), B functor }\} \\
&= \mathsf{l} \cdot (\mathsf{id}\times !) && \{\ \tau_r \text{ unit (C.1) }\} \\
&= \pi_1 && \{\ \pi_1 \text{ definition again }\}
\end{aligned}$$

and similarly for the τ_l case.

□

Furthermore, should B also form a monad, then the following laws hold:

- monad axioms (recall laws (A.1) and (A.2)),

$$\mu \cdot \eta\mathsf{B} \;= \mu \cdot \mathsf{B}\eta \;=\; \mathsf{id} \tag{C.14}$$

$$\mu \cdot \mathsf{B}\mu \;= \mu \cdot \mu\mathsf{B} \tag{C.15}$$

and the natural laws for μ and η:

$$\mu \cdot \mathsf{BB}f = \mathsf{B}f \cdot \mu \qquad (C.16)$$

$$\eta \cdot f = \mathsf{B}f \cdot \eta \qquad (C.17)$$

- η and μ as strong natural transformations (recall *e.g.*, laws (A.6) and (A.7)):

$$\tau_r \cdot (\eta \times \mathsf{id}) = \eta \qquad (C.18)$$

$$\tau_r \cdot (\mu \times \mathsf{id}) = \mu \cdot \mathsf{B}\ \tau_r \cdot \tau_r \qquad (C.19)$$

$$\tau_l \cdot (\mathsf{id} \times \eta) = \eta \qquad (C.20)$$

$$\tau_l \cdot (\mathsf{id} \times \mu) = \mu \cdot \mathsf{B}\ \tau_l \cdot \tau_l \qquad (C.21)$$

- definition and properties of the distributive morphisms δ_r and δ_l for a strong monad (recall §A.10):

$$\delta_r = \tau_r \bullet \tau_l \qquad (C.22)$$

$$\delta_l = \tau_l \bullet \tau_r \qquad (C.23)$$

$$\delta_r \cdot \mathsf{s} = \mathsf{Bs} \cdot \delta_l \qquad (C.24)$$

$$\delta_r \cdot (\mathsf{id} \times \eta) = \tau_r \qquad (C.25)$$

$$\delta_r \cdot (\eta \times \mathsf{id}) = \tau_l \qquad (C.26)$$

$$\delta_l \cdot (\eta \times \mathsf{id}) = \tau_l \qquad (C.27)$$

$$\delta_l \cdot (\mathsf{id} \times \eta) = \tau_r \qquad (C.28)$$

$$\delta_r \cdot \mathsf{s} = \mathsf{Bs} \cdot \delta_l \qquad (C.29)$$

$$\delta_r \cdot (\mathsf{B}f \times \mathsf{B}g) = \mathsf{B}(f \times g) \cdot \delta_r \qquad (C.30)$$

$$\delta_l \cdot (\mathsf{B}f \times \mathsf{B}g) = \mathsf{B}(f \times g) \cdot \delta_l \qquad (C.31)$$

2. τ-Laws

3. Further useful results can be derived from the set of laws above. The lemmas which follow (colloquially referred to as the 'τ-laws') state some 'housekeeping' laws describing the effect of common context management isomorphisms in the presence of tensorial strength. In particular, we concentrate on associativity (a), commutativity (s), exchange (xl, xr and m) and distributivity (dl and dr). For the reader's convenience, we recall from chapter 2 the definitions of the exchange morphisms:

$$\begin{aligned} \mathsf{xl} &: A \times (B \times C) \longrightarrow B \times (A \times C) \\ &= \mathsf{a} \cdot (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{a}^\circ \qquad \text{(C.32)} \\ \mathsf{xr} &: A \times B \times C \longrightarrow A \times C \times B \\ &= \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a} \qquad \text{(C.33)} \\ \mathsf{m} &: (A \times B) \times (C \times D) \longrightarrow (A \times C) \times (B \times D) \\ &= \mathsf{a} \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{a}^\circ \qquad \text{(C.34)} \end{aligned}$$

4. LEMMA. Let B be a strong functor. Then,

$$\tau_r \cdot (\tau_l \times \mathsf{id}) \cdot \mathsf{a}^\circ \;=\; \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times \tau_r) \qquad \text{(C.35)}$$

holds, *cf.*,

$$\begin{tikzcd} A \times (\mathsf{B}W \times C) \arrow[r, "\mathsf{a}^\circ"] \arrow[d, "\mathsf{id} \times \tau_r"'] & (A \times \mathsf{B}W) \times C \arrow[d, "\tau_l \times \mathsf{id}"] \\ A \times \mathsf{B}(W \times C) \arrow[d, "\tau_l"'] & \mathsf{B}(A \times W) \times C \arrow[d, "\tau_r"] \\ \mathsf{B}(A \times (W \times C)) \arrow[r, "\mathsf{Ba}^\circ"] & \mathsf{B}((A \times W) \times C) \end{tikzcd}$$

Proof.

$$\begin{aligned} & \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot \mathsf{a}^\circ \\ = \quad & \{\ \tau_l, \tau_r \text{ interchange (C.11), } \times \text{ functor}\ \} \\ & \tau_r \cdot (\mathsf{Bs} \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\ = \quad & \{\ \mathsf{a} \text{ iso, } \tau_r \text{ natural (C.5)}\ \} \\ & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot \mathsf{a} \cdot (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\ = \quad & \{\ \tau_r \text{ associative (C.3)}\ \} \\ & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \tau_r \cdot \mathsf{a} \cdot (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\ = \quad & \{\ \mathsf{s} \text{ iso}\ \} \\ & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \tau_r \cdot (\mathsf{id} \times \mathsf{s}) \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a} \cdot (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\ = \quad & \{\ \tau_r \text{ natural (C.5)}\ \} \end{aligned}$$

$$
\begin{array}{cl}
 & \mathsf{B}(\mathsf{s}\times\mathsf{id})\cdot\mathsf{Ba}^{\circ}\cdot\mathsf{B}(\mathsf{id}\times\mathsf{s})\cdot\tau_r\cdot(\mathsf{id}\times\mathsf{s})\cdot\mathsf{a}\cdot(\mathsf{s}\times\mathsf{id})\cdot\mathsf{a}^{\circ}\\
= & \qquad\{\ \mathsf{B}\text{ functor, routine: }(\mathsf{s}\times\mathsf{id})\cdot\mathsf{a}^{\circ}\cdot(\mathsf{id}\times\mathsf{s})=\mathsf{a}^{\circ}\cdot\mathsf{s}\cdot\mathsf{a}^{\circ}\ \}\\
 & \mathsf{Ba}^{\circ}\cdot\mathsf{Bs}\cdot\mathsf{Ba}^{\circ}\cdot\tau_r\cdot(\mathsf{id}\times\mathsf{s})\cdot\mathsf{a}\cdot(\mathsf{s}\times\mathsf{id})\cdot\mathsf{a}^{\circ}\\
= & \qquad\{\ \tau_r\text{ associative (C.3) }\}\\
 & \mathsf{Ba}^{\circ}\cdot\mathsf{Bs}\cdot\tau_r\cdot(\tau_r\times\mathsf{id})\cdot\mathsf{a}^{\circ}\cdot(\mathsf{id}\times\mathsf{s})\cdot\mathsf{a}\cdot(\mathsf{s}\times\mathsf{id})\cdot\mathsf{a}^{\circ}\\
= & \qquad\{\ \text{routine: }\mathsf{s}=\mathsf{a}^{\circ}\cdot(\mathsf{id}\times\mathsf{s})\cdot\mathsf{a}\cdot(\mathsf{s}\times\mathsf{id})\cdot\mathsf{a}^{\circ}\ \}\\
 & \mathsf{Ba}^{\circ}\cdot\mathsf{Bs}\cdot\tau_r\cdot(\tau_r\times\mathsf{id})\cdot\mathsf{s}\\
= & \qquad\{\ \mathsf{s}\text{ natural }\}\\
 & \mathsf{Ba}^{\circ}\cdot\mathsf{Bs}\cdot\tau_r\cdot\mathsf{s}\cdot(\mathsf{id}\times\tau_r)\\
= & \qquad\{\ \tau_l,\tau_r\text{ interchange (C.11) }\}\\
 & \mathsf{Ba}^{\circ}\cdot\tau_l\cdot(\mathsf{id}\times\tau_r)
\end{array}
$$

□

5. ANIMATION. To build up one's intuitions, CHARITY can be used to 'animate' these laws. We have proceeded in this way when trying to identify some useful results, before carrying out the formal proofs. The following printout is an example of 'running' law (C.35) proved above. The *sequence* monad implementation, given in §A.13, is used.

```
>> taur fp{taul,fid} iassoc ([5,3], ([1,2,3], 9)).
[((([5, 3], 1), 9), ((([5, 3], 2), 9), ((([5, 3], 3), 9)]
 : list((list(int) * int) * int)

>> B{iassoc} taul fp{fid,taur} ([5,3], ([1,2,3], 9)).
[((([5, 3], 1), 9), ((([5, 3], 2), 9), ((([5, 3], 3), 9)]
 : list((list(int) * int) * int)
```

6. LEMMA. Let B be a strong functor. Then,

$$\tau_r\cdot(\tau_l\times\mathsf{id})\cdot\mathsf{s}\ =\mathsf{Bs}\cdot\tau_l\cdot(\mathsf{id}\times\tau_l) \qquad \text{(C.36)}$$

$$\tau_r\cdot(\tau_r\times\mathsf{id})\cdot\mathsf{s}\ =\mathsf{Bs}\cdot\tau_l\cdot(\mathsf{id}\times\tau_r) \qquad \text{(C.37)}$$

Proof.

$$\begin{aligned}
\tau_r \cdot (\tau_l \times \mathsf{id}) \cdot \mathsf{s} &= \tau_r \cdot \mathsf{s} \cdot (\mathsf{id} \times \tau_l) && \{ \text{ s natural } \} \\
&= \mathsf{Bs} \cdot \tau_l \cdot (\mathsf{id} \times \tau_l) && \{ \tau_l, \tau_r \text{ interchange (C.7) } \}
\end{aligned}$$

proves (C.36); the proof of (C.37) follows an identical argument. The correspondent diagrams are:

$$\begin{tikzcd}
(A \times \mathsf{B}W) \times C \arrow[d, "\tau_l \times \mathsf{id}"'] & C \times (A \times \mathsf{B}W) \arrow[l, "\mathsf{s}"'] \arrow[d, "\mathsf{id} \times \tau_l"] \\
\mathsf{B}(A \times W) \times C \arrow[d, "\tau_r"'] & C \times \mathsf{B}(A \times W) \arrow[d, "\tau_l"] \\
\mathsf{B}((A \times W) \times C) & \mathsf{B}(C \times (A \times W)) \arrow[l, "\mathsf{Bs}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
(\mathsf{B}W \times A) \times C \arrow[d, "\tau_r \times \mathsf{id}"'] & C \times (\mathsf{B}W \times A) \arrow[l, "\mathsf{s}"'] \arrow[d, "\mathsf{id} \times \tau_r"] \\
\mathsf{B}(W \times A) \times C \arrow[d, "\tau_r"'] & C \times \mathsf{B}(W \times A) \arrow[d, "\tau_l"] \\
\mathsf{B}((W \times A) \times C) & \mathsf{B}(C \times (W \times A)) \arrow[l, "\mathsf{Bs}"']
\end{tikzcd}$$

□

7. LEMMA. Let B be a strong functor. Then,

$$\begin{aligned}
\mathsf{Bxr} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) &= \tau_l \cdot \mathsf{xr} && \text{(C.38)} \\
\mathsf{Bxr} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) &= \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{xr} && \text{(C.39)} \\
\mathsf{Bxl} \cdot \tau_l \cdot (\mathsf{id} \times \tau_r) &= \tau_r \cdot \mathsf{xl} && \text{(C.40)} \\
\mathsf{Bxl} \cdot \tau_l \cdot (\mathsf{id} \times \tau_l) &= \tau_l \cdot (\mathsf{id} \times \tau_l) \cdot \mathsf{xl} && \text{(C.41)}
\end{aligned}$$

Proof. We prove below xr equations (C.38) and (C.39). The corresponding laws for xl follow a similar argument. Equation (C.38) expresses the commutativity of the following diagram:

$$\begin{tikzcd}
(A \times \mathsf{B}B) \times C \arrow[r, "\tau_l \times \mathsf{id}"] \arrow[d, "\mathsf{xr}"'] & \mathsf{B}(A \times B) \times C \arrow[r, "\tau_r"] & \mathsf{B}((A \times B) \times C) \arrow[d, "\mathsf{Bxr}"'] \\
(A \times C) \times \mathsf{B}B \arrow[rr, "\tau_l"] & & \mathsf{B}((A \times C) \times B)
\end{tikzcd}$$

This leads to the following reasoning:

$$\begin{aligned}
& \mathsf{Bxr} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \\
= \quad & \quad \{ \ \mathsf{xr} \text{ definition (C.33)} \ \} \\
& \mathsf{Ba}^\circ \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \\
= \quad & \quad \{ \ \text{law (C.46)} \ \} \\
& \mathsf{Ba}^\circ \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \\
= \quad & \quad \{ \ \tau_r \text{ natural (C.5)} \ \} \\
& \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times \mathsf{Bs}) \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \\
= \quad & \quad \{ \ \tau_l, \tau_r \text{ interchangeable (C.7)} \ \} \\
& \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times \tau_l) \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a} \\
= \quad & \quad \{ \ \text{law (C.9)} \ \} \\
& \tau_l \cdot \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a} \\
= \quad & \quad \{ \ \mathsf{xr} \text{ definition (C.33)} \ \} \\
& \tau_l \cdot \mathsf{xr}
\end{aligned}$$

Now consider equation (C.39) expressing the commutativity of

$$\begin{tikzcd}
(\mathsf{B}A \times B) \times C \arrow[r, "\mathsf{xr}"] \arrow[d, "\tau_r \times \mathsf{id}"'] & (\mathsf{B}A \times C) \times B \arrow[r, "\tau_r \times \mathsf{id}"] & \mathsf{B}(A \times C) \times B \arrow[d, "\tau_r"] \\
\mathsf{B}(A \times B) \times C \arrow[r, "\tau_r"] & \mathsf{B}((A \times B) \times C) \arrow[r, "\mathsf{Bxr}"] & \mathsf{B}((A \times C) \times B)
\end{tikzcd}$$

Thus,

$$\begin{aligned}
& \mathsf{Bxr} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \\
= \quad & \quad \{ \ \mathsf{B} \text{ functor, } \mathsf{xr} \text{ definition (C.33)} \ \} \\
& \mathsf{Ba}^\circ \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \\
= \quad & \quad \{ \ \tau_r \text{ associative (C.8)} \ \} \\
& \mathsf{Ba}^\circ \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \tau_r \cdot \mathsf{a} \\
= \quad & \quad \{ \ \tau_r \text{ natural (C.5)} \ \} \\
& \mathsf{Ba}^\circ \cdot \tau_r \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a} \\
= \quad & \quad \{ \ \tau_r \text{ associative (C.3)} \ \} \\
& \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a} \\
= \quad & \quad \{ \ \mathsf{xr} \text{ definition (C.33)} \ \} \\
& \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

□

8. LEMMA. Let B be a strong functor. Then,

$$\begin{aligned}
\mathsf{Bm} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) &= \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{m} && \text{(C.42)}\\
\mathsf{Bm} \cdot \tau_l \cdot (\tau_l \times \mathsf{id}) &= \tau_l \cdot (\tau_l \times \mathsf{id}) \cdot \mathsf{m} && \text{(C.43)}\\
\mathsf{Bm} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) &= \tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{m} && \text{(C.44)}\\
\mathsf{Bm} \cdot \tau_l \cdot (\mathsf{id} \times \tau_r) &= \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot \mathsf{m} && \text{(C.45)}
\end{aligned}$$

Proof. Consider, first, equation (C.42) which express the commutativity of the following diagram (the proof of (C.43) follows a similar pattern):

$$\begin{tikzcd}
(\mathsf{B}W \times A) \times (E \times F) \arrow[r, "\mathsf{m}"] \arrow[d, "\tau_r \times \mathsf{id}"'] & (\mathsf{B}W \times E) \times (A \times F) \arrow[d, "\tau_r \times \mathsf{id}"] \\
\mathsf{B}(W \times A) \times (E \times F) \arrow[d, "\tau_r"'] & \mathsf{B}(W \times E) \times (A \times F) \arrow[d, "\tau_r"] \\
\mathsf{B}((W \times A) \times (E \times F)) \arrow[r, "\mathsf{Bm}"'] & \mathsf{B}((W \times E) \times (A \times F))
\end{tikzcd}$$

Thus,

$$\tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{m}$$

$$= \quad \{ \times \text{ functor, } \mathsf{m} \text{ definition (C.34)} \}$$

$$\tau_r \cdot (\tau_r \times (\mathsf{id} \times \mathsf{id})) \cdot \mathsf{a} \cdot ((\mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a}) \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$$= \quad \{ \mathsf{a} \text{ natural} \}$$

$$\tau_r \cdot \mathsf{a} \cdot ((\tau_r \times \mathsf{id}) \times \mathsf{id}) \cdot ((\mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a}) \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$$= \quad \{ \tau_r \text{ associative (C.8)} \}$$

$$\mathsf{Ba} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot ((\tau_r \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{a}^\circ \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$$= \quad \{ \tau_r \text{ associative (C.3)} \}$$

$$\mathsf{Ba} \cdot \tau_r \cdot (\mathsf{Ba}^\circ \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$$= \quad \{ \tau_r \text{ natural (C.5)} \}$$

$$\mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$$= \quad \{ \tau_r \text{ natural (C.5) again} \}$$

$$\mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$= \quad \{\ \tau_r$ associative (C.8) $\}$

$$\mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{Ba} \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((\tau_r \times \mathsf{id}) \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$= \quad \{\ \tau_r$ natural (C.5) $\}$

$$\mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id})$$
$$\cdot((\tau_r \times \mathsf{id}) \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$= \quad \{\ \mathsf{a}^\circ$ natural $\}$

$$\mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\tau_r \times (\mathsf{id} \times \mathsf{id}))$$

$= \quad \{$ functors, τ_r associative (C.3) $\}$

$$\mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \tau_r \cdot (\tau_r \times \mathsf{id})$$

$= \quad \{\ \mathsf{B}$ functor, m definition (C.34) $\}$

$$\mathsf{Bm} \cdot \tau_r \cdot (\tau_r \times \mathsf{id})$$

Next consider equation (C.44) which involves both τ_r and τ_l, expressing the commutativity of the diagram below. Again, the proof of (C.45) follows a similar pattern.

$$\begin{tikzcd}
(A \times \mathsf{B}W) \times (E \times F) \arrow[r, "\mathsf{m}"] \arrow[d, "\tau_l \times \mathsf{id}"'] & (A \times E) \times (\mathsf{B}W \times F) \arrow[d, "\mathsf{id} \times \tau_r"] \\
\mathsf{B}(A \times W) \times (E \times F) \arrow[d, "\tau_r"'] & (A \times E) \times \mathsf{B}(W \times F) \arrow[d, "\tau_l"] \\
\mathsf{B}((A \times W) \times (E \times F)) \arrow[r, "\mathsf{Bm}"'] & \mathsf{B}((A \times E) \times (W \times F))
\end{tikzcd}$$

Thus,

$$\tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{m}$$

$= \quad \{\ \mathsf{m}$ definition (C.34) $\}$

$$\tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \cdot ((\mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{s}) \cdot \mathsf{a}) \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$= \quad \{$ law (C.46), $\times$ functor $\}$

$$\mathsf{Ba} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{a}^\circ \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$= \quad \{\ \tau_l$ associative (C.9) $\}$

$$\mathsf{Ba} \cdot \tau_r \cdot (\mathsf{Ba}^\circ \times \mathsf{id}) \cdot (\tau_l \times \mathsf{id}) \cdot ((\mathsf{id} \times \tau_l) \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$= \quad \{\ \tau_r$ natural (C.5) $\}$

$$\mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot ((\mathsf{id} \times \tau_l) \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ$$

$= \quad \{\ \tau_r, \tau_l$ interchangeable (C.7) $\}$

$$\begin{array}{ll}
 & \mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{Bs}) \times \mathsf{id}) \cdot ((\mathsf{id} \times \tau_r) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\
= & \quad \{\ \tau_l \text{ natural (C.6)}\ \} \\
 & \mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot (\tau_l \times \mathsf{id}) \cdot ((\mathsf{id} \times \tau_r) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\
= & \quad \{\ \tau_r \text{ natural (C.5)}\ \} \\
 & \mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot ((\mathsf{id} \times \tau_r) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^\circ \\
= & \quad \{\ \text{law (C.46)}\ \} \\
 & \mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{Ba} \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((\tau_l \times \mathsf{id}) \times \mathsf{id}) \cdot \mathsf{a}^\circ \\
= & \quad \{\ \tau_r \text{ natural (C.5)}\ \} \\
 & \mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot ((\tau_l \times \mathsf{id}) \times \mathsf{id}) \cdot \mathsf{a}^\circ \\
= & \quad \{\ \mathsf{a}^\circ \text{ natural, functors}\ \} \\
 & \mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\tau_l \times \mathsf{id}) \\
= & \quad \{\ \tau_r \text{ associative (C.3)}\ \} \\
 & \mathsf{Ba} \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}((\mathsf{id} \times \mathsf{s}) \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \\
= & \quad \{\ \mathsf{m} \text{ definition (C.34)}\ \} \\
 & \mathsf{Bm} \cdot \tau_r \cdot (\tau_l \times \mathsf{id})
\end{array}$$

□

9. REMARK. Alternative formulations of laws in §§4, 6 and 8 are easily obtained by pre- or post-composing both sides of the corresponding equations with relevant isomorphisms. Useful variants of (C.35), (C.36) and (C.37) are, respectively,

$$\tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \;=\; \mathsf{Ba} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \tag{C.46}$$

$$\mathsf{Bs} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \;=\; \tau_l \cdot (\mathsf{id} \times \tau_l) \cdot \mathsf{s} \tag{C.47}$$

$$\mathsf{Bs} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \;=\; \tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{s} \tag{C.48}$$

Variants of laws involving xr, xl and m in the previous paragraphs are also easy to grasp.

10. DISTRIBUTIVITY. Distributivity plays a key role in defining and reasoning about some component combinators. As with other ‘housekeeping’ morphisms, we basically rely on the fact that both dr and dl are natural isomorphisms and sometimes use the explicit definition of their inverses. The following equalities, relating distributivity with the additive units, l_+ and r_+, and the *either* constructor, have been singled out. Consider first the following commutative diagrams:

$$\begin{tikzcd}
A \times (B + C) \arrow[d, "\mathsf{dr}"'] \arrow[dr, "f \times [h_1, h_2]"] & \\
A \times B + A \times C \arrow[r, "{[f \times h_1, f \times h_2]}"'] & D \times E
\end{tikzcd}
\qquad
\begin{tikzcd}
A \times (B + \emptyset) \arrow[r, "\mathsf{id} \times \mathsf{l}_+"] \arrow[d, "\mathsf{dr}"'] & A \times B \\
A \times B + A \times \emptyset \arrow[r, "\mathsf{id} + \mathsf{zr}"'] & A \times B + \emptyset \arrow[u, "\mathsf{l}_+"']
\end{tikzcd}$$

i.e.,

$$f \times [h_1, h_2] = [f \times h_1, f \times h_2] \cdot \mathsf{dr} \tag{C.49}$$

$$\mathsf{id} \times \mathsf{l}_+ = \mathsf{l}_+ \cdot (\mathsf{id} + \mathsf{zr}) \cdot \mathsf{dr} \tag{C.50}$$

Proof. For equation (C.49) consider

$$\begin{aligned}
& f \times [h_1, h_2] \;=\; [f \times h_1, f \times h_2] \cdot \mathsf{dr} \\
\equiv \quad & \{ \; \mathsf{dr} \text{ iso} \; \} \\
& f \times [h_1, h_2] \cdot \mathsf{dr}^\circ \;=\; [f \times h_1, f \times h_2] \\
\equiv \quad & \{ \; \mathsf{dr}^\circ \text{ definition, } + \text{ fusion} \; \} \\
& [(f \times [h_1, h_2]) \cdot (\mathsf{id} \times \iota_1), (f \times [h_1, h_2]) \cdot (\mathsf{id} \times \iota_2)] \;=\; [f \times h_1, f \times h_2] \\
\equiv \quad & \{ \; + \text{ cancellation} \; \} \\
& [f \times h_1, f \times h_2] \;=\; [f \times h_1, f \times h_2]
\end{aligned}$$

Concerning (C.50):

$$\begin{aligned}
& \mathsf{id} \times \mathsf{l}_+ \;=\; \mathsf{l}_+ \cdot (\mathsf{id} + \mathsf{zr}) \cdot \mathsf{dr} \\
\equiv \quad & \{ \; \mathsf{dr} \text{ iso} \; \} \\
& (\mathsf{id} \times \mathsf{l}_+) \cdot \mathsf{dr}^\circ \;=\; \mathsf{l}_+ \cdot (\mathsf{id} + \mathsf{zr}) \\
\equiv \quad & \{ \; \mathsf{dr}^\circ \text{ and } \mathsf{l}_+ \text{ definition} \; \} \\
& (\mathsf{id} \times [\mathsf{id}, ?]) \cdot [\mathsf{id} \times \iota_1, \mathsf{id} \times \iota_2] \;=\; [\mathsf{id}, ?] \cdot (\mathsf{id} + \mathsf{zr}) \\
\equiv \quad & \{ \; + \text{ fusion, cancellation, absorption} \; \} \\
& [\mathsf{id} \times \mathsf{id}, \mathsf{id} \times ?] \;=\; [\mathsf{id}, ? \cdot \mathsf{zr}] \\
\equiv \quad & \{ \; \text{functors, initiality } (\textit{cf.}, ?_{A \times B} \cdot \mathsf{zr}_A = \mathsf{id}_A \times ?_B) \; \} \\
& [\mathsf{id}, ? \cdot \mathsf{zr}] \;=\; [\mathsf{id}, ? \cdot \mathsf{zr}]
\end{aligned}$$

□

Similarly, for left distribution

$$\begin{aligned}[h_1,h_2]\times f &= [h_1\times f, h_2\times f]\cdot \mathsf{dl} && \text{(C.51)}\\ \mathsf{l}_+\times \mathsf{id} &= \mathsf{l}_+\cdot(\mathsf{id}+\mathsf{zl})\cdot\mathsf{dl} && \text{(C.52)}\end{aligned}$$

Both equalities (C.50) and (C.52) have ‘twin versions’ for r_+ replacing l_+:

$$\begin{aligned}\mathsf{id}\times\mathsf{r}_+ &= \mathsf{r}_+\cdot(\mathsf{zr}+\mathsf{id})\cdot\mathsf{dr} && \text{(C.53)}\\ \mathsf{r}_+\times\mathsf{id} &= \mathsf{r}_+\cdot(\mathsf{zl}+\mathsf{id})\cdot\mathsf{dl} && \text{(C.54)}\end{aligned}$$

The following equalities are very useful and trivially proved:

$$\begin{aligned}\iota_1 &= \mathsf{dr}\cdot(\mathsf{id}\times\iota_1) && \text{(C.55)}\\ \iota_2 &= \mathsf{dr}\cdot(\mathsf{id}\times\iota_2) && \text{(C.56)}\\ \iota_1 &= \mathsf{dl}\cdot(\iota_1\times\mathsf{id}) && \text{(C.57)}\\ \iota_2 &= \mathsf{dl}\cdot(\iota_2\times\mathsf{id}) && \text{(C.58)}\\ \mathsf{dr} &= \iota_1\cdot(\mathsf{id}\times\mathsf{l}_+) && \text{(C.59)}\\ \mathsf{dr} &= \iota_2\cdot(\mathsf{id}\times\mathsf{r}_+) && \text{(C.60)}\\ \mathsf{dl} &= \iota_1\cdot(\mathsf{l}_+\times\mathsf{id}) && \text{(C.61)}\\ \mathsf{dl} &= \iota_2\cdot(\mathsf{r}_+\times\mathsf{id}) && \text{(C.62)}\end{aligned}$$

Proof. As an example let us prove (C.59):

$$\begin{aligned}& \mathsf{dr} = \iota_1\cdot(\mathsf{id}\times\mathsf{l}_+)\\ \equiv\quad & \{\ \mathsf{dr}^\circ\ \text{iso}\ \}\\ & \mathsf{dr}^\circ\cdot\mathsf{dr} = \mathsf{dr}^\circ\cdot\iota_1\cdot(\mathsf{id}\times\mathsf{l}_+)\\ \equiv\quad & \{\ \mathsf{dr}\ \text{iso, law (C.55)}\ \}\\ & \mathsf{id} = (\mathsf{id}\times\iota_1)\cdot(\mathsf{id}\times\mathsf{l}_+)\\ \equiv\quad & \{\ \text{routine: } \iota_1\cdot\mathsf{l}_+ = \mathsf{id},\ \times\ \text{functor}\ \}\\ & \mathsf{id} = \mathsf{id}\end{aligned}$$

□

Finally, observe that successive applications of dr° (or dl°) are often used to simplify expressions. For example, $[\mathsf{id}\times(\mathsf{id}\times\iota_1),\mathsf{id}\times(\mathsf{id}\times\iota_2)]$ can be used to transform a sum $(U\times(V\times A))+(U\times(V\times B))$ into $U\times(V\times(A+B))$. A simple calculation, resorting to $+$ fusion and cancellation, establishes

$$[\mathsf{id}\times(\mathsf{id}\times\iota_1),\mathsf{id}\times(\mathsf{id}\times\iota_2)] \;=\; (\mathsf{id}\times\mathsf{dr}^\circ)\cdot\mathsf{dr}^\circ \qquad \text{(C.63)}$$

The next two lemmas relate the distributivity morphisms to monad unit η and strength.

11. LEMMA. Let B be a monad. Then,

$$\eta \cdot \mathsf{dr}^\circ = [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\eta + \eta) \tag{C.64}$$

$$\eta \cdot \mathsf{dl}^\circ = [\mathsf{B}(\iota_1 \times \mathsf{id}), \mathsf{B}(\iota_2 \times \mathsf{id})] \cdot (\eta + \eta) \tag{C.65}$$

Proof.

$$\begin{array}{ll} & \eta \cdot \mathsf{dr}^\circ \\ = & \qquad \{ \ \mathsf{dr}^\circ \text{ definition } \} \\ & \eta \cdot [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)] \\ = & \qquad \{ \ + \text{ fusion } \} \\ & [\eta \cdot (\mathsf{id} \times \iota_1), \eta \cdot (\mathsf{id} \times \iota_2)] \\ = & \qquad \{ \ \eta \text{ natural (C.17) } \} \\ & [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \eta, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \eta] \\ = & \qquad \{ \ + \text{ absorption } \} \\ & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\eta + \eta) \end{array}$$

The proof of (C.65) is analogous.

□

12. LEMMA. Let B be a strong functor. Then,

$$\begin{array}{l} \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \ = \\ \qquad [\mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id})] \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \end{array} \tag{C.66}$$

$$\begin{array}{l} \tau_l \cdot (\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \ = \\ \qquad [\mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_1)), \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_2))] \cdot (\tau_l + \tau_l) \cdot \mathsf{dr} \end{array} \tag{C.67}$$

Proof. For the proof of (C.66) consider the post-composition of both sides of the equation with dl°. Thus,

$$
\begin{array}{ll}
 & \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot \mathsf{dl}^\circ \\
= & \quad \{ \ \mathsf{dl}^\circ \text{ definition } \} \\
 & \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot [\iota_1 \times \mathsf{id}, \iota_2 \times \mathsf{id}] \\
= & \quad \{ \ + \text{ fusion and cancellation } \} \\
 & \tau_r \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \times \mathsf{id}, \mathsf{B}(\mathsf{id} \times \iota_2) \times \mathsf{id}] \\
= & \quad \{ \ + \text{ fusion again } \} \\
 & [\tau_r \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \times \mathsf{id}, \tau_r \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \times \mathsf{id}] \\
= & \quad \{ \ \tau_r \text{ natural (C.5) } \} \\
 & [\mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}) \cdot \tau_r, \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id}) \cdot \tau_r] \\
= & \quad \{ \ + \text{ absorption } \} \\
 & [\mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id})] \cdot (\tau_r + \tau_r)
\end{array}
$$

The proof of (C.67) is analogous.

□

3. δ-Laws

13. LEMMA. Let B be a strong monad. Then,

$$\delta \cdot (\eta \times \eta) \ = \eta \tag{C.68}$$

where δ is either δ_r or δ_l.

Proof. Let us consider the equality for $\delta = \delta_l$. The case $\delta = \delta_r$ is proved in a similar way.

$$
\begin{array}{ll}
 & \delta_l \cdot (\eta \times \eta) \\
= & \quad \{ \ \delta_l \text{ definition (C.23) } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\eta \times \eta) \\
= & \quad \{ \ \times \text{ functor } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\eta \times \mathsf{id}) \cdot (\mathsf{id} \times \eta) \\
= & \quad \{ \ \eta \text{ strong (C.18) } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \eta \cdot (\mathsf{id} \times \eta) \\
= & \quad \{ \ \eta \text{ natural (C.17) } \} \\
 & \mu \cdot \eta \cdot \tau_l \cdot (\mathsf{id} \times \eta) \\
= & \quad \{ \ \mu \text{ unit (C.14) } \}
\end{array}
$$

$$
\begin{array}{ll}
 & \tau_l \cdot (\mathsf{id} \times \eta) \\
= & \quad \{\ \eta \text{ strong (C.20) } \} \\
 & \eta
\end{array}
$$

□

14. LEMMA. Let B be a strong monad. Then,

$$\delta_r \bullet \tau_r = \tau_r \bullet \delta_l \quad \textit{i.e.}, \mu \cdot \mathsf{B}\delta_r \cdot \tau_r = \mu \cdot \mathsf{B}\tau_r \cdot \delta_l \tag{C.69}$$

$$\delta_l \bullet \tau_l = \tau_l \bullet \delta_r \quad \textit{i.e.}, \mu \cdot \mathsf{B}\delta_l \cdot \tau_l = \mu \cdot \mathsf{B}\tau_l \cdot \delta_r \tag{C.70}$$

Proof. We prove (C.69) below; law (C.70) is verified by a similar argument.

$$
\begin{array}{ll}
 & \mu \cdot \mathsf{B}\delta_r \cdot \tau_r \\
= & \quad \{\ \delta_r \text{ definition (C.22) } \} \\
 & \mu \cdot \mathsf{B}\mu \cdot \mathsf{BB}\tau_r \cdot \mathsf{B}\tau_l \cdot \tau_r \\
= & \quad \{\ \mu \text{ associative (C.15) } \} \\
 & \mu \cdot \mu \cdot \mathsf{BB}\tau_r \cdot \mathsf{B}\tau_l \cdot \tau_r \\
= & \quad \{\ \mu \text{ natural (C.16) } \} \\
 & \mu \cdot \mathsf{B}\tau_r \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \\
= & \quad \{\ \delta_l \text{ definition (C.23) } \} \\
 & \mu \cdot \mathsf{B}\tau_r \cdot \delta_l
\end{array}
$$

□

15. LEMMA. Let B be a strong monad. Then,

$$\tau_r \bullet \delta_r = \mu \cdot \mathsf{B}\tau_r \cdot \delta_r = \delta_r \cdot (\mu \times \mathsf{id}) \tag{C.71}$$

$$\tau_l \bullet \delta_l = \mu \cdot \mathsf{B}\tau_l \cdot \delta_l = \delta_l \cdot (\mathsf{id} \times \mu) \tag{C.72}$$

$$\delta_r \bullet \tau_l = \mu \cdot \mathsf{B}\delta_r \cdot \tau_l = \delta_r \cdot (\mathsf{id} \times \mu) \tag{C.73}$$

$$\delta_l \bullet \tau_r = \mu \cdot \mathsf{B}\delta_l \cdot \tau_r = \delta_l \cdot (\mu \times \mathsf{id}) \tag{C.74}$$

Proof. We prove (C.71) and (C.73). The proofs of (C.72) and (C.74) follow, respectively, similar arguments. Thus,

$$
\begin{array}{ll}
 & \mu \cdot \mathsf{B}\tau_r \cdot \delta_r \\
= & \qquad \{ \ \delta_r \text{ definition (C.22)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_r \cdot \mathsf{B}\mu \cdot \mathsf{B}\tau_r \cdot \tau_l \\
= & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
 & \mu \cdot \mu \cdot \mathsf{BB}\tau_r \cdot \mathsf{B}\tau_r \cdot \tau_l \\
= & \qquad \{ \ \mu \text{ associative (C.15)} \ \} \\
 & \mu \cdot \mathsf{B}\mu \cdot \mathsf{BB}\tau_r \cdot \mathsf{B}\tau_r \cdot \tau_l \\
= & \qquad \{ \ \mu \text{ strong (C.19)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\mu \times \mathsf{id}) \cdot \tau_l \\
= & \qquad \{ \ \tau_l \text{ natural (C.6)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_r \cdot \tau_l \cdot (\mu \times \mathsf{id}) \\
= & \qquad \{ \ \delta_r \text{ definition (C.22)} \ \} \\
 & \delta_r \cdot (\mu \times \mathsf{id})
\end{array}
$$

Now (C.73) is established as follows:

$$
\begin{array}{ll}
 & \mu \cdot \mathsf{B}\delta_r \cdot \tau_l \\
= & \qquad \{ \ \delta_r \text{ definition (C.22)} \ \} \\
 & \mu \cdot \mathsf{B}\mu \cdot \mathsf{BB}\tau_r \cdot \mathsf{B}\tau_l \cdot \tau_l \\
= & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_r \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_l \\
= & \qquad \{ \ \mu \text{ strong (C.21)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_r \cdot \tau_l \cdot (\mathsf{id} \times \mu) \\
= & \qquad \{ \ \delta_r \text{ definition (C.22)} \ \} \\
 & \mu \cdot \delta_r \cdot (\mathsf{id} \times \mu)
\end{array}
$$

□

16. REMARK. Laws (C.71) and (C.74) in the previous lemma state the commutativity of the following diagrams:

$$\begin{tikzcd}
\mathsf{BB}A \times \mathsf{B}C \arrow[r, "\delta_r"] \arrow[d, "(\mu \times \mathsf{id})"'] & \mathsf{B}(\mathsf{B}A \times C) \arrow[r, "\mathsf{B}\tau_r"] & \mathsf{BB}(A \times C) \arrow[d, "\mu"] \\
\mathsf{B}A \times \mathsf{B}C \arrow[rr, "\delta_r"'] & & \mathsf{B}(A \times C)
\end{tikzcd}$$

and

$$\begin{tikzcd}
\mathsf{BB}A \times \mathsf{B}C \arrow[r, "\tau_r"] \arrow[d, "(\mu \times \mathsf{id})"'] & \mathsf{B}(\mathsf{B}A \times \mathsf{B}C) \arrow[r, "\mathsf{B}\delta_l"] & \mathsf{BB}(A \times C) \arrow[d, "\mu"] \\
\mathsf{B}A \times \mathsf{B}C \arrow[rr, "\delta_l"'] & & \mathsf{B}(A \times C)
\end{tikzcd}$$

Similar diagrams can be drawn for the other two equalities. Curiously, however, the 'more symmetric' diagram below does not commute no matter how each occurrence of δ is replaced by either δ_r or δ_l.

$$\begin{tikzcd}
\mathsf{BB}A \times \mathsf{BB}C \arrow[r, "\delta"] \arrow[d, "(\mu \times \mu)"'] & \mathsf{B}(\mathsf{B}A \times \mathsf{B}C) \arrow[r, "\mathsf{B}\delta"] & \mathsf{BB}(A \times C) \arrow[d, "\mu"] \\
\mathsf{B}A \times \mathsf{B}C \arrow[rr, "\delta"'] & & \mathsf{B}(A \times C)
\end{tikzcd}$$

In fact, the diagram only commutes under the additional hypothesis that B is a commutative monad (§A.10). In this case $\delta_r = \delta_l$ and the result follows as detailed in the next paragraph.

17. LEMMA. Let B be a commutative strong monad. Then,

$$\delta \bullet \delta \; = \mu \cdot \mathsf{B}\delta \cdot \delta \; = \delta \cdot (\mu \times \mu) \qquad \text{(C.75)}$$

where $\delta = \delta_r = \delta_l$.

Proof.

$$\begin{array}{ll}
& \mu \cdot \mathsf{B}\delta \cdot \delta \\
= & \qquad \{ \text{ instantiating } \delta \; \} \\
& \mu \cdot \mathsf{B}\delta_r \cdot \delta_l \\
= & \qquad \{ \; \delta_l \text{ definition (C.23) } \} \\
& \mu \cdot \mathsf{B}\delta_r \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \\
= & \qquad \{ \; \mu \text{ natural (C.16) } \} \\
& \mu \cdot \mu \cdot \mathsf{BB}\delta_r \cdot \mathsf{B}\tau_l \cdot \tau_r
\end{array}$$

$$
\begin{aligned}
= &\quad \{\ \mu \text{ associative (C.15)}\ \} \\
&\mu \cdot \mathsf{B}\mu \cdot \mathsf{BB}\delta_r \cdot \mathsf{B}\tau_l \cdot \tau_r \\
= &\quad \{\ \text{law (C.73)}\ \} \\
&\mu \cdot \mathsf{B}\delta_r \cdot \mathsf{B}(\mathsf{id} \times \mu) \cdot \tau_r \\
= &\quad \{\ \tau_r \text{ natural (C.5)}\ \} \\
&\mu \cdot \mathsf{B}\delta_r \cdot \tau_r \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \text{law (C.69)}\ \} \\
&\mu \cdot \mathsf{B}\tau_r \cdot \delta_l \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \text{assumption: } \delta_r = \delta_l\ \} \\
&\mu \cdot \mathsf{B}\tau_r \cdot \delta_r \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \delta_r \text{ definition (C.22)}\ \} \\
&\mu \cdot \mathsf{B}\tau_r \cdot \mu \cdot \mathsf{B}\tau_r \cdot \tau_l \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \mu \text{ natural (C.16)}\ \} \\
&\mu \cdot \mu \cdot \mathsf{BB}\tau_r \cdot \mathsf{B}\tau_r \cdot \tau_l \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \mu \text{ associative (C.15)}\ \} \\
&\mu \cdot \mathsf{B}\mu \cdot \mathsf{BB}\tau_r \cdot \mathsf{B}\tau_r \cdot \tau_l \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \mu \text{ strong (C.19)}\ \} \\
&\mu \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\mu \times \mathsf{id}) \cdot \tau_l \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \tau_l \text{ natural (C.6)}\ \} \\
&\mu \cdot \mathsf{B}\tau_r \cdot \tau_l \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{id} \times \mu) \\
= &\quad \{\ \delta_r \text{ definition (C.22), recovering } \delta \text{ as } \delta_r\ \} \\
&\delta \cdot (\mu \times \mu)
\end{aligned}
$$

□

18. ANIMATION. A CHARITY animation of the results above provides some intuition about why and when commutativity is required. Resorting to the implementations of both the *sequence* and *maybe* monads discussed in §A.13, consider first law (C.72). The examples in the following printout suggest the law is valid for both monads, irrespective of the former (but not the latter) being non commutative. Thus, for the *sequence* monad one gets, for instance

```
>> mu B{taul} deltal ([1,2],[[4,7],[9]]).
[(1, 4), (1, 7), (1, 9), (2, 4), (2, 7), (2, 9)] : list(int * int)

>> deltal fp{fid,mu} ([1,2],[[4,7],[9]]).
[(1, 4), (1, 7), (1, 9), (2, 4), (2, 7), (2, 9)] : list(int * int)
```

and, for the *maybe* monad,

```
>> mu B{taul} deltal (ss 6, ss ss 8).
ss(6, 8) : SF(int * int)

>> deltal fp{fid,mu} (ss 6, ss ss 8).
ss(6, 8) : SF(int * int)

>> mu B{taul} deltal (ss 6, ss ff).
ff : SF(int * A)

>> deltal fp{fid,mu} (ss 6, ss ff).
ff : SF(int * A)
```

The situation is completely different when dealing with law (C.75). In the animation below we have taken δ as δ_l. The application of both sides of the equation to concrete examples leads to identical results in case the monad is commutative. Such is not the case if commutativity fails, as in the *sequence* monad case:

```
>> mu B{deltal} deltal ([[1,2],[4]],[[5,6],[8,7]]).
[(1, 5), (1, 6), (2, 5), (2, 6), (1, 8), (1, 7), (2, 8), (2, 7),
 (4, 5), (4, 6), (4, 8), (4, 7)] : list(int * int)

>> deltal fp{mu,mu} ([[1,2],[4]],[[5,6],[8,7]]).
[(1, 5), (1, 6), (1, 8), (1, 7), (2, 5), (2, 6), (2, 8), (2, 7),
 (4, 5), (4, 6), (4, 8), (4, 7)] : list(int * int)
```

19. LEMMA. Let B be a strong monad. Then,

$$\begin{aligned} \mathsf{Bs} \cdot \delta_r &= \delta_l \cdot \mathsf{s} && \text{(C.76)} \\ \mathsf{Ba} \cdot \delta_l \cdot (\tau_l \times \mathsf{id}) &= \tau_l \cdot (\mathsf{id} \times \delta_l) \cdot \mathsf{a} && \text{(C.77)} \\ \delta_r \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} &= \mathsf{Ba} \cdot \tau_r \cdot (\delta_r \times \mathsf{id}) && \text{(C.78)} \end{aligned}$$

Proof. Equation (C.76) establishes δ_r, δ_l interchangeablity, which stems from a similar law for τ_r and τ_l:

$$
\begin{array}{ll}
 & \mathsf{Bs} \cdot \delta_r \\
\equiv & \qquad \{ \ \delta_r \text{ definition (C.22)} \ \} \\
 & \mathsf{Bs} \cdot \mu \cdot \mathsf{B}\tau_r \cdot \tau_l \\
\equiv & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
 & \mu \cdot \mathsf{BBs} \cdot \mathsf{B}\tau_r \cdot \tau_l \\
\equiv & \qquad \{ \ \tau_r, \tau_l \text{ interchangeablity (C.7)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{Bs} \cdot \tau_l \\
\equiv & \qquad \{ \ \tau_r, \tau_l \text{ interchangeablity (C.7) again} \ \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot \mathsf{s} \\
\equiv & \qquad \{ \ \delta_l \text{ definition (C.23)} \ \} \\
 & \delta_l \cdot \mathsf{s}
\end{array}
$$

Let us now check (C.77), the proof of (C.78) being similar. Of course, both equalities can also be formulated in terms of a°, for example:

$$
\begin{array}{ll}
 & \delta_r \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \ = \ \mathsf{Ba} \cdot \tau_r \cdot (\delta_r \times \mathsf{id}) \\
\equiv & \qquad \{ \ \text{composition with isomorphisms} \ \} \\
 & \mathsf{Ba}^\circ \cdot \delta_r \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \cdot \mathsf{a}^\circ \ = \ \mathsf{Ba}^\circ \cdot \mathsf{Ba} \cdot \tau_r \cdot (\delta_r \times \mathsf{id}) \cdot \mathsf{a}^\circ \\
\equiv & \qquad \{ \ \mathsf{a} \cdot \mathsf{a}^\circ = \mathsf{id} \} \\
 & \mathsf{Ba}^\circ \cdot \delta_r \cdot (\mathsf{id} \times \tau_r) \ = \ \tau_r \cdot (\delta_r \times \mathsf{id}) \cdot \mathsf{a}^\circ
\end{array}
$$

Thus

$$
\begin{array}{ll}
 & \mathsf{Ba} \cdot \delta_l \cdot (\tau_l \times \mathsf{id}) \\
= & \qquad \{ \ \delta_l \text{ definition (C.23)} \} \\
 & \mathsf{Ba} \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \\
= & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
 & \mu \cdot \mathsf{BBa} \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \\
= & \qquad \{ \ \tau_l \text{ associativity (C.4)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \\
= & \qquad \{ \ \text{law (C.46)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \\
= & \qquad \{ \ \tau_l \text{ natural (C.6)} \ \}
\end{array}
$$

$$\begin{aligned}
& \mu \cdot \mathsf{B}\tau_l \cdot \tau_l \cdot (\mathsf{id} \times \mathsf{B}\tau_l) \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \\
= & \qquad \{ \ \mu \text{ strong (C.21)} \ \} \\
& \tau_l \cdot (\mathsf{id} \times \mu) \cdot (\mathsf{id} \times \mathsf{B}\tau_l) \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \\
= & \qquad \{ \ \delta_l \text{ definition (C.23)} \ \} \\
& \tau_l \cdot (\mathsf{id} \times \delta_l) \cdot \mathsf{a}
\end{aligned}$$

□

20. LEMMA. Let B be a strong monad. Then,

$$\mathsf{Ba} \cdot \delta \cdot (\delta \times \mathsf{id}) \ = \delta \cdot (\mathsf{id} \times \delta) \cdot \mathsf{a} \tag{C.79}$$

where $\delta = \delta_r$ or $\delta = \delta_l$.

Proof. The law establishes the commutativity of the following diagram, where δ stands either for δ_r or δ_l.

$$\begin{tikzcd}
(\mathsf{B}A \times \mathsf{B}C) \times \mathsf{B}D \arrow[r, "\mathsf{a}"] \arrow[d, "\delta \times \mathsf{id}"'] & \mathsf{B}A \times (\mathsf{B}C \times \mathsf{B}D) \arrow[d, "\mathsf{id} \times \delta"] \\
\mathsf{B}(A \times C) \times \mathsf{B}D \arrow[d, "\delta"'] & \mathsf{B}A \times \mathsf{B}(C \times D) \arrow[d, "\delta"] \\
\mathsf{B}((A \times C) \times D) \arrow[r, "\mathsf{Ba}"'] & \mathsf{B}(A \times (C \times D))
\end{tikzcd}$$

Let us prove the δ_l version of this law (the δ_r case is similar).

$$\begin{aligned}
& \mathsf{Ba} \cdot \delta_l \cdot (\delta_l \times \mathsf{id}) \\
= & \qquad \{ \ \delta_l \text{ definition (C.23)} \ \} \\
& \mathsf{Ba} \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\delta_l \times \mathsf{id}) \\
= & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
& \mu \cdot \mathsf{BBa} \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\delta_l \times \mathsf{id}) \\
= & \qquad \{ \ \tau_l \text{ associative (C.4)} \ \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\delta_l \times \mathsf{id}) \\
= & \qquad \{ \ \delta_l \text{ definition (C.23)} \ \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \\
= & \qquad \{ \ \mu \text{ strong (C.19)} \ \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mathsf{Ba} \cdot \mu \cdot \mathsf{B}\tau_r \cdot \tau_r \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \\
= & \qquad \{ \ \mu \text{ natural (C.16)}, \tau_r \text{ natural (C.5)} \ \}
\end{aligned}$$

$$
\begin{aligned}
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mu \cdot \mathsf{BBa} \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \\
= \quad & \qquad \{ \text{ law (C.46) } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \\
= \quad & \qquad \{ \ \tau_r \text{ associative (C.8) } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \tau_r \cdot \mathsf{a} \\
= \quad & \qquad \{ \ \mu \text{ natural (C.16) } \} \\
& \mu \cdot \mu \cdot \mathsf{BB}\tau_l \cdot \mathsf{BB}(\mathsf{id} \times \tau_l) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \tau_r \cdot \mathsf{a} \\
= \quad & \qquad \{ \ \tau_l \text{ natural (C.6) } \} \\
& \mu \cdot \mu \cdot \mathsf{BB}\tau_l \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}\tau_l) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \tau_r \cdot \mathsf{a} \\
= \quad & \qquad \{ \ \mu \text{ associative (C.15) } \} \\
& \mu \cdot \mathsf{B}\mu \cdot \mathsf{BB}\tau_l \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}\tau_l) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \tau_r \cdot \mathsf{a} \\
= \quad & \qquad \{ \ \mu \text{ strong (C.21) } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mu) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}\tau_l) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \tau_r \cdot \mathsf{a} \\
= \quad & \qquad \{ \ \tau_r \text{ natural (C.5) } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\mathsf{id} \times \mu) \cdot (\mathsf{id} \times \mathsf{B}\tau_l) \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \\
= \quad & \qquad \{ \ \delta_l \text{ definition (C.23) } \} \\
& \delta_l \cdot (\mathsf{id} \times \delta_l) \cdot \mathsf{a}
\end{aligned}
$$

□

21. LEMMA. Let B be a strong monad. Then,

$$\delta \cdot (\tau_r \times \tau_r) \cdot \mathsf{m} \ = \mathsf{Bm} \cdot \tau_r \cdot (\delta \times \mathsf{id}) \qquad \text{(C.80)}$$

$$\delta \cdot (\tau_l \times \tau_l) \cdot \mathsf{m} \ = \mathsf{Bm} \cdot \tau_l \cdot (\mathsf{id} \times \delta) \qquad \text{(C.81)}$$

where δ is either δ_r or δ_l.

Proof. The laws express the commutativity of the following diagrams:

$$\begin{tikzcd}
\mathsf{B}(A \times B) \times \mathsf{B}(C \times D) \times (E \times F) \arrow[r, "\mathsf{m}"] \arrow[d, "\delta \times \mathsf{id}"'] & (\mathsf{B}(A \times B) \times E) \times (\mathsf{B}(C \times D) \times F) \arrow[d, "\tau_r \times \tau_r"] \\
\mathsf{B}((A \times B) \times (C \times D)) \times (E \times F) \arrow[d, "\tau_r"'] & \mathsf{B}(A \times B \times E) \times \mathsf{B}(C \times D \times F) \arrow[d, "\delta"] \\
\mathsf{B}((A \times B) \times (C \times D) \times (E \times F)) \arrow[r, "\mathsf{Bm}"] & \mathsf{B}((A \times B \times E) \times (C \times D \times F))
\end{tikzcd}$$

and

$$\begin{tikzcd}
(A \times B) \times (\mathsf{B}(C \times D) \times \mathsf{B}(E \times F)) \arrow[r, "\mathsf{m}"] \arrow[d, "\mathsf{id} \times \delta"'] & (A \times \mathsf{B}(C \times D)) \times (B \times \mathsf{B}(E \times F)) \arrow[d, "\tau_l \times \tau_l"] \\
(A \times B) \times (\mathsf{B}(C \times D) \times (E \times F)) \arrow[d, "\tau_l"'] & \mathsf{B}(A \times (C \times D)) \times \mathsf{B}(B \times (E \times F)) \arrow[d, "\delta"] \\
\mathsf{B}((A \times B) \times ((C \times D) \times (E \times F)) \arrow[r, "\mathsf{Bm}"] & \mathsf{B}((A \times (C \times D)) \times (B \times (E \times F)))
\end{tikzcd}$$

Consider equality (C.80) for $\delta = \delta_l$. The remaining cases are similar.

$$\begin{array}{cl}
 & \mathsf{Bm} \cdot \tau_r \cdot (\delta_l \times \mathsf{id}) \\
= & \quad \{ \ \delta_l \text{ definition (C.23)} \ \} \\
 & \mathsf{Bm} \cdot \tau_r \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \\
= & \quad \{ \ \mu \text{ strong (C.19)} \ \} \\
 & \mathsf{Bm} \cdot \mu \cdot \mathsf{B}\tau_r \cdot \tau_r \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \\
= & \quad \{ \ \tau_r \text{ natural (C.5)} \ \} \\
 & \mathsf{Bm} \cdot \mu \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \\
= & \quad \{ \ \mu \text{ natural (C.16)} \ \} \\
 & \mu \cdot \mathsf{BBm} \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \\
= & \quad \{ \ \text{law (C.44), } \mathsf{B} \text{ functor} \ \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Bm} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \\
= & \quad \{ \ \text{law (C.42)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{m} \\
= & \quad \{ \ \tau_r \text{ natural (C.5)} \ \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (\mathsf{id} \times \tau_r) \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{m} \\
= & \quad \{ \ \delta_l \text{ definition, } \times \text{ functor} \ \} \\
 & \delta_l \cdot (\tau_r \times \tau_r) \cdot \mathsf{m}
\end{array}$$

□

APPENDIX D

Proofs

1. Proofs for Chapter 4

LEMMA §4.13

Proof. (4.1)

$$
\begin{array}{cl}
 & \omega \cdot + \cdot (+ \times \mathsf{id}) \\
\equiv & \qquad \{ \ + \text{ definition } \} \\
 & \cup \cdot (\omega \times \omega) \cdot (+ \times \mathsf{id}) \\
\equiv & \qquad \{ \ \times \text{ functor } \} \\
 & \cup \cdot (\omega \cdot + \times \omega) \\
\equiv & \qquad \{ \ + \text{ definition } \} \\
 & \cup \cdot (\cup \cdot (\omega \times \omega)) \times \omega \\
\equiv & \qquad \{ \ \times \text{ functor } \} \\
 & \cup \cdot (\cup \times \mathsf{id}) \cdot ((\omega \times \omega) \cdot \omega) \\
\equiv & \qquad \{ \ \mathsf{a} \text{ isomorphism } \} \\
 & \cup \cdot (\cup \times \mathsf{id}) \cdot ((\omega \times \omega) \cdot \omega) \cdot \mathsf{a}^{\circ} \cdot \mathsf{a} \\
\equiv & \qquad \{ \ \mathsf{a}^{\circ} \text{ natural } \} \\
 & \cup \cdot (\cup \times \mathsf{id}) \cdot \mathsf{a}^{\circ} \cdot (\omega \times (\omega \times \omega)) \cdot \mathsf{a} \\
\equiv & \qquad \{ \ \cup \text{ associative } \} \\
 & \cup \cdot (\mathsf{id} \times \cup) \cdot (\omega \times (\omega \times \omega)) \cdot \mathsf{a} \\
\equiv & \qquad \{ \ \times \text{ functor } \} \\
 & \cup \cdot (\omega \times (\cup \cdot (\omega \times \omega)) \cdot \mathsf{a} \\
\equiv & \qquad \{ \ + \text{ definition } \}
\end{array}
$$

$$
\begin{aligned}
& \cup \cdot (\omega \times \omega \cdot +) \cdot \mathsf{a} \\
\equiv & \qquad \{ \times \text{ functor} \} \\
& \cup \cdot (\omega \times \omega) \cdot (\mathsf{id} \times +) \cdot \mathsf{a} \\
\equiv & \qquad \{ + \text{ definition} \} \\
& \omega \cdot + \cdot (\mathsf{id} \times +) \cdot \mathsf{a}
\end{aligned}
$$

(4.2)

$$
\begin{aligned}
& \omega \cdot + \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ} \\
\equiv & \qquad \{ + \text{ definition} \} \\
& \cup \cdot (\omega \times \omega) \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ} \\
\equiv & \qquad \{ \times \text{ functor} \} \\
& \cup \cdot (\omega \cdot \mathsf{id} \times \omega \cdot \mathsf{nil}) \cdot \mathsf{l}^{\circ} \\
\equiv & \qquad \{ \mathsf{nil} \text{ definition} \} \\
& \cup \cdot (\omega \cdot \underline{\emptyset}) \cdot \mathsf{l}^{\circ} \\
\equiv & \qquad \{ \mathsf{l}^{\circ} \text{ definition} \} \\
& \cup \cdot (\omega \cdot \underline{\emptyset}) \cdot \langle \mathsf{id}, ! \rangle \\
\equiv & \qquad \{ \times \text{ absorption} \} \\
& \cup \cdot \langle \omega \cdot \mathsf{id}, \underline{\emptyset} \cdot ! \rangle \\
\equiv & \qquad \{ \cup \text{ identity} \} \\
& \omega
\end{aligned}
$$

(4.3)

$$
\begin{aligned}
& \omega \cdot + \cdot \, \triangle \\
\equiv & \qquad \{ + \text{ definition} \} \\
& \cup \cdot (\omega \times \omega) \cdot \, \triangle \\
\equiv & \qquad \{ \triangle \text{ natural} \} \\
& \cup \cdot \, \triangle \cdot \omega \\
\equiv & \qquad \{ \cup \text{ idempotent} \} \\
& \omega
\end{aligned}
$$

□

$\boxed{\text{LEMMA §4.16}}$

Proof. Associativity and unit laws remain to be proved. The last case is detailed below; the proof of associativity follows a similar argument. We want to verify that nil is a unit for $|||$, *i.e.*, $b\ |||\ \mathsf{nil} = b$, or, going pointfree, $||| \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ = \mathsf{id}$. Therefore,

$$
\begin{array}{cl}
 & ||| \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ \ = \ \mathsf{id} \\
\equiv & \qquad \{ \text{ ana reflection (3.8) } \} \\
 & ||| \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ \ = \ [\![(\omega)]\!] \\
\Leftarrow & \qquad \{ \text{ ana universal (3.6) } \} \\
 & \omega \cdot ||| \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ \ = \ \mathcal{P}(\mathsf{id} \times ||| \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ) \cdot \omega
\end{array}
$$

which holds because

$$
\begin{array}{cl}
 & \omega \cdot ||| \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ \\
= & \qquad \{ \text{ comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_{|||} \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \alpha_{|||} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \triangle \text{ natural } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{nil}) \times (\mathsf{id} \times \omega \cdot \mathsf{nil})) \cdot \triangle \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \times \text{ functor and } \mathsf{nil} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \cdot (\omega \times \mathsf{nil}) \times \tau_l \cdot (\mathsf{id} \times \underline{\emptyset})) \cdot \triangle \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \tau_l \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \cdot (\omega \times \mathsf{nil}) \times \underline{\emptyset}) \cdot \triangle \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \cup \text{ unit } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \tau_r \cdot (\omega \times \mathsf{nil}) \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \times \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \tau_r \cdot (\mathsf{id} \times \mathsf{nil}) \cdot (\omega \times \mathsf{id}) \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \tau_r \text{ natural (C.5) } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \mathcal{P}(\mathsf{id} \times (\mathsf{id} \times \mathsf{nil})) \cdot \tau_r \cdot (\omega \times \mathsf{id}) \cdot \mathsf{l}^\circ \\
= & \qquad \{ \ \star \text{ below } \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \mathcal{P}(\mathsf{id} \times (\mathsf{id} \times \mathsf{nil})) \cdot \mathcal{P}(\mathsf{id} \times \mathsf{l}^\circ) \cdot \omega \cdot \mathsf{l} \cdot \mathsf{l}^\circ
\end{array}
$$

$$
\begin{aligned}
= \quad & \{\ \text{functors, } \mathsf{l} \text{ isomorphism}\ \} \\
& \mathcal{P}(\mathsf{id} \times ||| \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ}) \cdot \omega
\end{aligned}
$$

Step $\star$ is justified as folows

$$
\begin{aligned}
& \tau_r \cdot (\omega \times \mathsf{id}) \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{l}^{\circ}) \cdot \omega \cdot \mathsf{l} \\
\equiv \quad & \{\ \mathsf{l} \text{ iso}\ \} \\
& \mathcal{P}(\mathsf{id} \times \mathsf{l}) \cdot \tau_r \cdot (\omega \times \mathsf{id}) \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{l}) \cdot \mathcal{P}(\mathsf{id} \times \mathsf{l}^{\circ}) \cdot \omega \cdot \mathsf{l} \\
\equiv \quad & \{\ \mathsf{l} \text{ natural iso}\ \} \\
& \mathcal{P}(\mathsf{id} \times \mathsf{l}) \cdot \tau_r \cdot \mathsf{l}^{\circ} \cdot \omega \cdot \mathsf{l} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{l}) \cdot \mathcal{P}(\mathsf{id} \times \mathsf{l}^{\circ}) \cdot \omega \cdot \mathsf{l} \\
\equiv \quad & \{\ \text{law (C.1)}\ \} \\
& \mathsf{l} \cdot \mathsf{l}^{\circ} \cdot \omega \cdot \mathsf{l} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{l} \cdot \mathsf{l}^{\circ}) \cdot \omega \cdot \mathsf{l} \\
\equiv \quad & \{\ \mathsf{l} \text{ iso, functors}\ \} \\
& \mathsf{id} \cdot \omega \cdot \mathsf{l} \;=\; \mathsf{id} \cdot \omega \cdot \mathsf{l}
\end{aligned}
$$

□

LEMMA §4.18

Proof. Consider, first, equation (4.5):

$$\begin{array}{cl}
 & \omega \cdot \backslash_K \cdot + \\
= & \quad \{ \text{ comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K} \cdot + \\
= & \quad \{ \ \alpha_{\backslash_K} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \cdot + \\
= & \quad \{ \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \cup \cdot (\omega \times \omega) \\
= & \quad \{ \ \mathsf{filter}_K \text{ commutes with } \cup, \times \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \cup \cdot (\mathsf{filter}_K \cdot \omega \times \mathsf{filter}_K \cdot \omega) \\
= & \quad \{ \ \alpha_{\backslash_K} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \cup \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= & \quad \{ \ \cup \text{ natural } \} \\
 & \cup \cdot (\mathcal{P}(\mathsf{id} \times \backslash_K) \times \mathcal{P}(\mathsf{id} \times \backslash_K))(\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= & \quad \{ \text{ comorphism}, \times \text{ functor } \} \\
 & \cup \cdot (\omega \cdot \backslash_K \times \omega \cdot \backslash_K) \\
= & \quad \{ \ \times \text{ functor } \} \\
 & \cup \cdot (\omega \times \omega) \cdot (\backslash_K \times \backslash_K) \\
= & \quad \{ \ + \text{ definition } \} \\
 & \omega \cdot + \cdot (\backslash_K \times \backslash_K)
\end{array}$$

To establish equation (4.6), we shall prove that $\omega \cdot \backslash_K \cdot |||$ and $\omega \cdot ||| \cdot (\backslash_K \cdot \backslash_K)$ unfold, respectively, to $\mathcal{P}(\mathsf{id} \times \backslash_K \cdot |||) \cdot \alpha$ and $\mathcal{P}(\mathsf{id} \times ||| \cdot (\backslash_K \times \backslash_K) \cdot \alpha$, for

$$\alpha \ = \ \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K}))\cdot \triangle$$

Unfolding the first expression yields

$$\begin{array}{cl}
 & \omega \cdot \backslash_K \cdot ||| \\
= & \quad \{ \text{ comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K} \cdot ||| \\
= & \quad \{ \ \alpha_{\backslash_K} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \cdot |||
\end{array}$$

$$
\begin{array}{ll}
= & \quad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_{|||} \\
= & \quad \{ \ \mathsf{filter}_K \text{ natural } \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \mathsf{filter}_K \cdot \alpha_{|||} \\
= & \quad \{ \ \alpha_{|||} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |||)) \cdot \mathsf{filter}_K \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \\
= & \quad \{ \ \mathsf{filter}_K \text{ commutes with } \cup \ \} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |||)) \cdot \cup \cdot (\mathsf{filter}_K \times \mathsf{filter}_K) \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \\
= & \quad \{ \ \tau_r, \tau_l \text{ natural (C.5) and (C.6) } \} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |||)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\mathsf{filter}_K \times \mathsf{id}) \times (\mathsf{id} \times \mathsf{filter}_K)) \\
& \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \\
= & \quad \{ \ \times \text{ functor } \} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |||)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\mathsf{filter}_K \cdot \omega \times \mathsf{id}) \times (\mathsf{id} \times \mathsf{filter}_K \cdot \omega)) \cdot \triangle \\
= & \quad \{ \ \alpha_{|||} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot |||)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K})) \cdot \triangle
\end{array}
$$

The second expression, on its turn, unfolds as follows:

$$
\begin{array}{ll}
& \omega \cdot ||| \cdot (\backslash_K \cdot \backslash_K) \\
= & \quad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_{|||} \cdot (\backslash_K \times \backslash_K) \\
= & \quad \{ \ \alpha_{|||} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \cdot (\backslash_K \times \backslash_K) \\
= & \quad \{ \ \triangle \text{ natural, } \times \text{ functor } \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \cdot \backslash_K \times \backslash_K) \times (\backslash_K \times \omega \cdot \backslash_K)) \cdot \triangle \\
= & \quad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \\
& \cdot ((\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K} \times \backslash_K) \times (\backslash_K \times \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K})) \cdot \triangle \\
= & \quad \{ \ \times \text{ functor } \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\mathcal{P}(\mathsf{id} \times \backslash_K) \times \backslash_K) \times (\backslash_K \times \mathcal{P}(\mathsf{id} \times \backslash_K))) \\
& \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K})) \cdot \triangle
\end{array}
$$

$$
\begin{array}{ll}
= & \quad \{\ \tau_r, \tau_l \text{ natural (C.5) and (C.6)} \ \} \\
 & \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \times \mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K))) \cdot (\tau_r \times \tau_l) \\
 & \cdot((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K}))\cdot \triangle \\
= & \quad \{\ \text{union} \ \} \\
 & \mathcal{P}(\mathsf{id} \times ||| \cdot (\backslash_K \times \backslash_K)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K}))\cdot \triangle
\end{array}
$$

Finally, equation (4.7):

$$
\begin{array}{ll}
 & \omega \cdot \backslash_K \cdot a. \\
= & \quad \{\ \text{comorphism} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K} \cdot a. \\
= & \quad \{\ \alpha_{\backslash_K} \text{ definition} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \cdot a. \\
= & \quad \{\ \textit{prefix} \text{ definition} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathsf{sing} \cdot \mathsf{label}_a \\
= & \quad \{\ \mathsf{filter}_K \text{ properties} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot (a \in K \ \rightarrow\ \emptyset, \mathsf{sing} \cdot \mathsf{label}_a) \\
= & \quad \{\ \text{conditional (2.41)} \ \} \\
 & (a \in K \ \rightarrow\ \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \underline{\emptyset}, \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{sing} \cdot \mathsf{label}_a) \\
= & \quad \{\ \mathsf{sing} \cdot \mathsf{label}_a \text{ natural, powerset} \ \} \\
 & (a \in K \ \rightarrow\ \underline{\emptyset}, \mathsf{sing} \cdot \mathsf{label}_a \cdot \backslash_K) \\
= & \quad \{\ \textit{prefix} \text{ definition} \ \} \\
 & (a \in K \ \rightarrow\ \omega \cdot \mathsf{nil}, \omega \cdot a. \cdot \backslash_K) \\
= & \quad \{\ \text{conditional (2.41)} \ \} \\
 & \omega \cdot (a \in K \ \rightarrow\ \mathsf{nil}, a. \cdot \backslash_K)
\end{array}
$$

□

LEMMA §4.19

Proof. Equation (4.8) is established by

$$\begin{array}{ll}
 & \backslash_K \cdot \backslash_K \;=\; \backslash_K \\
\equiv & \qquad \{\ \alpha_{\backslash_K} \text{ definition }\} \\
 & [\![(\alpha_{\backslash_K})]\!] \cdot \backslash_K \;=\; [\![(\alpha_{\backslash_K})]\!] \\
\Leftarrow & \qquad \{\ \text{ana fusion (3.9) }\} \\
 & \alpha_{\backslash_K} \cdot \backslash_K \;=\; \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K}
\end{array}$$

where the last step is justified by the following calculation

$$\begin{array}{ll}
 & \alpha_{\backslash_K} \cdot \backslash_K \\
\equiv & \qquad \{\ \alpha_{\backslash_K} \text{ definition }\} \\
 & \mathsf{filter}_K \cdot \omega \cdot \backslash_K \\
\equiv & \qquad \{\ \text{comorphism }\} \\
 & \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K} \\
\equiv & \qquad \{\ \mathsf{filter}_K \text{ natural }\} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \alpha_{\backslash_K} \\
\equiv & \qquad \{\ \alpha_{\backslash_K} \text{ definition }\} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathsf{filter}_K \cdot \omega \\
\equiv & \qquad \{\ \mathsf{filter}_K \text{ idempotency }\} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \\
\equiv & \qquad \{\ \alpha_{\backslash_K} \text{ definition }\} \\
 & \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \alpha_{\backslash_K}
\end{array}$$

□

LEMMA §4.23

Proof. For equation (4.9), compute

$$\begin{aligned}
& \omega \cdot [\mathsf{id}] \\
= & \qquad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times [\mathsf{id}]) \cdot \alpha_{[\mathsf{id}]} \\
= & \qquad \{ \ \alpha_{[\mathsf{id}]} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [\mathsf{id}]) \cdot \mathcal{P}(\mathsf{id} \times \mathsf{id}) \cdot \omega \\
= & \qquad \{ \ \times \text{ functor } \} \\
& \mathcal{P}(\mathsf{id} \times [\mathsf{id}]) \cdot \omega
\end{aligned}$$

By ana fusion (3.9), this implies $\llbracket(\omega)\rrbracket \cdot [\mathsf{id}] = \llbracket(\omega)\rrbracket$, which equivales, by ana reflection (3.8), to $[\mathsf{id}] = \mathsf{id}$. To prove (4.10) note first that

$$\begin{aligned}
& \omega \cdot [f] \cdot [f'] \\
= & \qquad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]} \cdot [f'] \\
= & \qquad \{ \ \alpha_{[f]} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \omega \cdot [f'] \\
= & \qquad \{ \text{ comorphism and } \alpha_{[f]} \text{ definition again } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \mathcal{P}(\mathsf{id} \times [f']) \cdot \mathcal{P}(f' \times \mathsf{id}) \cdot \omega \\
= & \qquad \{ \ \times \text{ functor } \} \\
& \mathcal{P}(\mathsf{id} \times [f] \cdot [f']) \cdot \mathcal{P}(f \cdot f' \times \mathsf{id}) \cdot \omega
\end{aligned}$$

On the other hand

$$\begin{aligned}
& \omega \cdot [f \cdot f'] \\
= & \qquad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times [f \cdot f']) \cdot \alpha_{[f \cdot f']} \\
= & \qquad \{ \ \alpha_{[f]} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [f] \cdot [f']) \cdot \mathcal{P}(f \cdot f' \times \mathsf{id}) \cdot \omega
\end{aligned}$$

and the result follows.

□

LEMMA §4.24

Proof.

(4.11)

$$
\begin{aligned}
& \omega \cdot [f] \cdot a. \\
= \quad & \quad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]} \cdot a. \\
= \quad & \quad \{ \ \alpha_{[f]} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \omega \cdot a. \\
= \quad & \quad \{ \ \textit{prefix} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \mathsf{sing} \cdot \mathsf{label}_a \\
= \quad & \quad \{ \ \mathsf{sing} \text{ natural } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathsf{sing} \cdot (f \times \mathsf{id}) \cdot \mathsf{label}_a \\
= \quad & \quad \{ \ \mathsf{label}_a \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathsf{sing} \cdot \mathsf{label}_{fa} \\
= \quad & \quad \{ \ \mathsf{sing} \text{ natural } \} \\
& \mathsf{sing} \cdot (\mathsf{id} \times [f]) \cdot \mathsf{label}_{fa} \\
= \quad & \quad \{ \ \mathsf{label}_a \text{ natural } \} \\
& \mathsf{sing} \cdot \mathsf{label}_{fa} \cdot [f] \\
= \quad & \quad \{ \ \textit{prefix} \text{ definition } \} \\
& \omega \cdot fa. \cdot [f]
\end{aligned}
$$

(4.12)

$$
\begin{aligned}
& \omega \cdot [f] \cdot + \\
= \quad & \quad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]} \cdot + \\
= \quad & \quad \{ \ \alpha_{[f]} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \omega \cdot + \\
= \quad & \quad \{ \ + \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \cup \cdot (\omega \times \omega) \\
= \quad & \quad \{ \ \cup \text{ natural and } \times \text{ functor } \}
\end{aligned}
$$

$$
\begin{aligned}
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \cup \cdot (\mathcal{P}(f \times \mathsf{id}) \cdot \omega \times \mathcal{P}(f \times \mathsf{id}) \cdot \omega) \\
= & \qquad \{ \ \cup \text{ natural} \ \} \\
& \cup \cdot (\mathcal{P}(\mathsf{id} \times [f]) \times \mathcal{P}(\mathsf{id} \times [f])) \cdot (\mathcal{P}(f \times \mathsf{id}) \cdot \omega \times \mathcal{P}(f \times \mathsf{id}) \cdot \omega) \\
= & \qquad \{ \ \times \text{ functor} \ \} \\
& \cup \cdot (\mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \omega \times \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \omega) \\
= & \qquad \{ \ \alpha_{[f]} \text{ definition} \ \} \\
& \cup \cdot (\mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]} \times \mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]}) \\
= & \qquad \{ \ \text{comorphism} \ \} \\
& \cup \cdot (\omega \cdot [f] \times \omega \cdot [f]) \\
= & \qquad \{ \ \times \text{ functor} \ \} \\
& \cup \cdot (\omega \times \omega) \cdot ([f] \times [f]) \\
= & \qquad \{ \ + \text{ definition} \ \} \\
& \omega \cdot + \cdot ([f] \times [f])
\end{aligned}
$$

(4.13) We first show that

$$
\begin{aligned}
& \omega \cdot [f] \cdot ||| \ = \\
= & \qquad \{ \ \text{comorphism} \ \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]} \cdot ||| \\
= & \qquad \{ \ \alpha_{[f]} \text{ definition} \ \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \omega \cdot ||| \\
= & \qquad \{ \ \text{comorphism} \ \} \\
& \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_{|||} \\
= & \qquad \{ \ \times \text{ functor and } \alpha_{|||} \text{ definition} \ \} \\
& \mathcal{P}(\mathsf{id} \times ([f] \cdot |||)) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \vartriangle \\
= & \qquad \{ \ \cup \text{ natural} \ \} \\
& \mathcal{P}(\mathsf{id} \times ([f] \cdot |||)) \cdot \cup \cdot (\mathcal{P}(f \times \mathsf{id}) \times \mathcal{P}(f \times \mathsf{id})) \cdot (\tau_r \times \tau_l) \\
& \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \vartriangle \\
= & \qquad \{ \ \tau_r, \tau_l \text{ natural (C.5) and (C.6)} \ \} \\
& \mathcal{P}(\mathsf{id} \times ([f] \cdot |||)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\mathcal{P}(f \times \mathsf{id}) \times \mathsf{id}) \times (\mathsf{id} \times \mathcal{P}(f \times \mathsf{id}))) \\
& \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \vartriangle \\
= & \qquad \{ \ \times \text{ functor} \ \}
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{P}(\mathsf{id} \times ([f] \cdot |||)) \cdot \cup \cdot (\tau_r \times \tau_l) \\
&\cdot((\mathcal{P}(f \times \mathsf{id}) \cdot \omega) \times \mathsf{id}) \times (\mathsf{id} \times (\mathcal{P}(f \times \mathsf{id}) \cdot \omega)))\cdot \Delta \\
= &\qquad \{\ \alpha_{[f]} \text{ definition}\ \} \\
&\mathcal{P}(\mathsf{id} \times ([f] \cdot |||)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{[f]} \times \mathsf{id}) \times (\mathsf{id} \times (\alpha_{[f]}))\cdot \Delta
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
&\omega \cdot ||| \cdot ([f] \times [f]) \\
= &\qquad \{\ \text{comorphism}\ \} \\
&\mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_{|||} \cdot ([f] \times [f]) \\
= &\qquad \{\ \alpha_{|||} \text{ definition}\ \} \\
&\mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega))\cdot \Delta \cdot([f] \times [f]) \\
= &\qquad \{\ \times \text{ functor and } \Delta \text{ natural}\ \} \\
&\mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \cdot [f] \times [f]) \times ([f] \times \omega \cdot [f]))\cdot \Delta \\
= &\qquad \{\ \alpha_{[f]} \text{ definition}\ \} \\
&\mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]} \times [f]) \times ([f] \times \mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]}))\cdot \Delta \\
= &\qquad \{\ \times \text{ functor}\ \} \\
&\mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\mathcal{P}(\mathsf{id} \times ([f] \times [f])) \times \mathcal{P}(\mathsf{id} \times ([f] \times [f]))) \cdot (\tau_r \times \tau_l) \\
&\cdot((\alpha_{[f]} \times \mathsf{id}) \times (\mathsf{id} \times (\alpha_{[f]}))\cdot \Delta \\
= &\qquad \{\ \cup \text{ natural}\ \} \\
&\mathcal{P}(\mathsf{id} \times |||) \cdot \mathcal{P}(\mathsf{id} \times ([f] \times [f])) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{[f]} \times \mathsf{id}) \times (\mathsf{id} \times (\alpha_{[f]}))\cdot \Delta \\
= &\qquad \{\ \times \text{ functor}\ \} \\
&\mathcal{P}(\mathsf{id} \times ||| \cdot ([f] \times [f])) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{[f]} \times \mathsf{id}) \times (\mathsf{id} \times (\alpha_{[f]}))\cdot \Delta
\end{aligned}
$$

□

LEMMA §4.30

Proof.

(4.15)

$$\begin{array}{ll}
 & \otimes \cdot \mathsf{s} \;=\; \otimes \\
\equiv & \qquad \{ \ \otimes \text{ definition } \} \\
 & [\![(\alpha_\otimes)]\!] \cdot \mathsf{s} \;=\; [\![(\alpha_\otimes)]\!] \\
\Leftarrow & \qquad \{ \ \text{ana fusion (3.9) } \} \\
 & \alpha_\otimes \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes \\
\equiv & \qquad \{ \ \alpha_\otimes \text{ definition } \} \\
 & \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes \\
\equiv & \qquad \{ \ \mathsf{s} \text{ natural } \} \\
 & \mathsf{sel} \cdot \delta_r \cdot \mathsf{s} \cdot (\omega \times \omega) \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes \\
\equiv & \qquad \{ \ \delta_r, \delta_l \text{ interaction (C.24) } \} \\
 & \mathsf{sel} \cdot \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \delta_l \cdot (\omega \times \omega) \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes \\
\equiv & \qquad \{ \ \mathsf{sel} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{sel} \cdot \delta_l \cdot (\omega \times \omega) \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes \\
\equiv & \qquad \{ \ \text{monad comutativity } \} \\
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{sel} \cdot \delta_r(\omega \times \omega) \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes \\
\equiv & \qquad \{ \ \alpha_\otimes \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes \;=\; \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_\otimes
\end{array}$$

(4.16) Once again we proceed by unfolding $\omega \cdot \otimes \cdot (\otimes \times \mathsf{id})$ and $\omega \cdot \otimes \cdot (\mathsf{id} \times \otimes) \cdot \mathsf{a}$ to identify a 'common' coalgebra. Thus,

$$\begin{array}{ll}
 & \omega \cdot \otimes \cdot (\otimes \times \mathsf{id}) \\
= & \qquad \{ \ \text{comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_\otimes \cdot (\otimes \times \mathsf{id}) \\
= & \qquad \{ \ \alpha_\otimes \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot (\otimes \times \mathsf{id}) \\
= & \qquad \{ \ \times \text{ functor, comorphism } \}
\end{array}$$

$$
\begin{array}{cl}
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(\mathsf{id} \times \otimes) \times \mathsf{id}) \cdot (\alpha_\otimes \times \omega) \\
= & \qquad \{ \ \delta_r \text{ natural (C.30)} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathcal{P}(\mathsf{id} \times (\otimes \times \mathsf{id})) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_\otimes \times \omega) \\
= & \qquad \{ \ \mathcal{P}(Act \times \mathsf{Id}) \text{ functor} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes \cdot (\otimes \times \mathsf{id})) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_\otimes \times \omega)
\end{array}
$$

and

$$
\begin{array}{cl}
 & \omega \cdot \otimes \cdot (\mathsf{id} \times \otimes) \cdot \mathsf{a} \\
= & \qquad \{ \ \text{comorphism} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_\otimes \cdot (\mathsf{id} \times \otimes) \cdot \mathsf{a} \\
= & \qquad \{ \ \alpha_\otimes \text{ definition} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot (\mathsf{id} \times \otimes) \cdot \mathsf{a} \\
= & \qquad \{ \ \times \text{ functor} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega \cdot \otimes) \cdot \mathsf{a} \\
= & \qquad \{ \ \times \text{ functor, comorphism} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(\mathsf{id} \times \otimes) \times \mathsf{id}) \cdot (\omega \times \alpha_\otimes) \cdot \mathsf{a} \\
= & \qquad \{ \ \mathsf{sel} \text{ definition, } \delta_r \text{ natural (C.30)} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathcal{P}(\mathsf{id} \times (\mathsf{id} \times \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \alpha_\otimes) \cdot \mathsf{a} \\
= & \qquad \{ \ \mathcal{P}(Act \times \mathsf{Id}) \text{ functor, } \alpha_\otimes \text{ definition} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes \cdot (\mathsf{id} \times \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega)) \cdot \mathsf{a} \\
= & \qquad \{ \ \times \text{ functor} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes \cdot (\mathsf{id} \times \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{id} \times \mathsf{sel} \cdot \delta_r) \cdot (\omega \times (\omega \times \omega)) \cdot \mathsf{a} \\
= & \qquad \{ \ \mathsf{a} \text{ natural} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes \cdot (\mathsf{id} \times \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{id} \times \mathsf{sel} \cdot \delta_r) \cdot \mathsf{a} \cdot ((\omega \times \omega) \times \omega) \\
= & \qquad \{ \ \mathsf{sel} \text{ definition, } \delta_r \text{ natural (C.30)} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes \cdot (\mathsf{id} \times \otimes)) \cdot \mathcal{P}(\mathsf{id} \times \mathsf{a}) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{sel} \cdot \delta_r \times \mathsf{id}) \cdot ((\omega \times \omega) \times \omega) \\
= & \qquad \{ \ \alpha_\otimes \text{ definition, } \mathcal{P}(Act \times \mathsf{Id}) \text{ functor} \ \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes \cdot (\mathsf{id} \times \otimes) \cdot \mathsf{a}) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_\otimes \cdot \omega)
\end{array}
$$

(4.17)

$$
\begin{array}{ll}
& \omega \cdot \otimes \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ} \\
= & \quad \{ \text{ comorphism } \} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_{\otimes} \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ} \\
= & \quad \{ \ \alpha_{\otimes} \text{ definition } \} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^{\circ} \\
= & \quad \{ \ \omega \cdot \mathsf{nil} = \underline{\emptyset}, \delta_r \text{ definition for monad } \mathcal{P}(Act \times \mathsf{Id}) \ \} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \underline{\emptyset} \cdot ! \cdot \mathsf{l}^{\circ} \\
= & \quad \{ \ \mathcal{P} \text{ and } \mathsf{sel} \text{ definition } \} \\
& \underline{\emptyset} \cdot ! \cdot \mathsf{l}^{\circ} \\
= & \quad \{ \text{ finality } \} \\
& \underline{\emptyset} \cdot ! \\
= & \quad \{ \ \mathsf{nil} \text{ definition } \} \\
& \omega \cdot \mathsf{nil} \cdot !
\end{array}
$$

□

LEMMA §4.31

Proof. The proof of (4.18) proceeds by showing that

$$\omega \cdot \otimes \cdot (\mathsf{id} \times +) \;=\; \omega \cdot + \cdot (\otimes \times \otimes) \cdot \mathsf{pdr}$$

Thus,

$$\begin{array}{ll}
 & \omega \cdot + \cdot (\otimes \times \otimes) \cdot \mathsf{pdr} \\
= & \qquad \{\ + \text{ definition } \} \\
 & \cup \cdot (\omega \times \omega) \cdot (\otimes \times \otimes) \cdot \mathsf{pdr} \\
= & \qquad \{\ \times \text{ functor } \} \\
 & \cup \cdot (\omega \cdot \otimes \times \omega \cdot \otimes) \cdot \mathsf{pdr} \\
= & \qquad \{\ \text{comorphism } \} \\
 & \cup \cdot (\mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_{\otimes} \times \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_{\otimes}) \cdot \mathsf{pdr} \\
= & \qquad \{\ \alpha_{\otimes} \text{ definition, } \times \text{ functor } \} \\
 & \cup \cdot (\mathcal{P}(\mathsf{id} \times \otimes) \times \mathcal{P}(\mathsf{id} \times \otimes)) \cdot (\mathsf{sel} \times \mathsf{sel}) \cdot (\delta_r \times \delta_r) \cdot ((\omega \times \omega) \times (\omega \times \omega)) \cdot \mathsf{pdr} \\
= & \qquad \{\ \cup \text{ natural } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \cup(\delta_r \times \delta_r) \cdot ((\omega \times \omega) \times (\omega \times \omega)) \cdot \mathsf{pdr} \\
= & \qquad \{\ \delta_r \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \cup \cdot (\mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \tau_l \times \mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \tau_l) \\
 & \cdot((\omega \times \omega) \times (\omega \times \omega)) \cdot \mathsf{pdr} \\
= & \qquad \{\ \cup \text{ natural } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \cup \cdot (\tau_l \times \tau_l) \cdot ((\omega \times \omega) \times (\omega \times \omega)) \cdot \mathsf{pdr} \\
= & \qquad \{\ \mathsf{pdr} \text{ definition, } \times \text{ absortion } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \cup \cdot (\tau_l \times \tau_l) \\
 & \cdot\langle (\omega \times \omega) \cdot (\mathsf{id} \times \pi_1), (\omega \times \omega) \cdot (\mathsf{id} \times \pi_2) \rangle \\
= & \qquad \{\ \times \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \cup \cdot (\tau_l \times \tau_l) \cdot \langle \omega \times \omega \cdot \pi_1, \omega \times \omega \cdot \pi_2 \rangle \\
= & \qquad \{\ \times \text{ cancellation } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \cup \cdot (\tau_l \times \tau_l) \cdot \langle \omega \times \pi_1 \cdot (\omega \times \omega), \omega \times \pi_2 \cdot (\omega \times \omega) \rangle \\
= & \qquad \{\ \mathsf{pdr} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \cup \cdot (\tau_l \times \tau_l) \cdot \mathsf{pdr} \cdot (\omega \times (\omega \times \omega)) \\
= & \qquad \{\ \cup \cdot (\tau_l \times \tau_l) \cdot \mathsf{pdr} \;=\; \tau_l \cdot (\mathsf{id} \times \cup) \ \}
\end{array}$$

$$
\begin{array}{ll}
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mu \cdot \mathcal{P}(\mathsf{id} \times \tau_r) \cdot \tau_l \cdot (\mathsf{id} \times \cup) \cdot (\omega \times (\omega \times \omega)) \\
= & \qquad \{\ \delta_r \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{id} \times \cup) \cdot (\omega \times (\omega \times \omega)) \\
= & \qquad \{\ + \text{ definition, } \times \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot (\mathsf{id} \times +) \\
= & \qquad \{\ \alpha_\otimes \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_\otimes \cdot (\mathsf{id} \times +) \\
= & \qquad \{\ \text{comorphism } \} \\
 & \omega \cdot \otimes \cdot (\mathsf{id} \times +)
\end{array}
$$

Concerning (4.20), we proceed by unfolding the two sides of the equation to identify a common coalgebra and, then, deriving the relevant side condition. Thus,

$$
\begin{array}{ll}
 & \omega \cdot \otimes \cdot ([f] \times [f]) \\
= & \qquad \{\ \text{comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_{[f]} \cdot ([f] \times [f]) \\
= & \qquad \{\ \alpha_{[f]} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot ([f] \times [f]) \\
= & \qquad \{\ \times \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \cdot [f] \times \omega \cdot [f]) \\
= & \qquad \{\ \text{comorphism } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]}) \times (\mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]}) \\
= & \qquad \{\ \times \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(\mathsf{id} \times [f]) \times \mathcal{P}(\mathsf{id} \times [f])) \cdot (\alpha_{[f]} \cdot \alpha_{[f]}) \\
= & \qquad \{\ \delta_r \text{ natural (C.30) } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mathcal{P}(\mathsf{id} \times ([f] \times [f])) \cdot \delta_r \cdot (\alpha_{[f]} \cdot \alpha_{[f]}) \\
= & \qquad \{\ \mathsf{sel} \text{ definition } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathcal{P}(\mathsf{id} \times ([f] \times [f])) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{[f]} \cdot \alpha_{[f]}) \\
= & \qquad \{\ \mathcal{P}(Act \times \mathsf{Id}) \text{ functor } \} \\
 & \mathcal{P}(\mathsf{id} \times \otimes \cdot ([f] \times [f]) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{[f]} \cdot \alpha_{[f]})
\end{array}
$$

On the other hand,

$$\omega \cdot [f] \cdot \otimes$$

$$
\begin{aligned}
&= \quad \{ \text{ comorphism } \}\\
&\quad \mathcal{P}(\mathsf{id} \times [f]) \cdot \alpha_{[f]} \cdot \otimes\\
&= \quad \{ \alpha_{[f]} \text{ definition } \}\\
&\quad \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \omega \cdot \otimes\\
&= \quad \{ \text{ comorphism } \}\\
&\quad \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_{\otimes}\\
&= \quad \{ \alpha_{\otimes} \text{ definition } \}\\
&\quad \mathcal{P}(\mathsf{id} \times [f]) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega)\\
&= \quad \{ \mathcal{P}(Act \times \mathsf{Id}) \text{ functor } \}\\
&\quad \mathcal{P}(\mathsf{id} \times ([f] \cdot \otimes)) \cdot \mathcal{P}(f \times \mathsf{id}) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega)\\
&\overset{?}{=} \quad \{ \star \}\\
&\quad \mathcal{P}(\mathsf{id} \times ([f] \cdot \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(f \times \mathsf{id}) \times \mathcal{P}(f \times \mathsf{id})) \cdot (\omega \times \omega)\\
&= \quad \{ \times \text{ functor } \}\\
&\quad \mathcal{P}(\mathsf{id} \times ([f] \cdot \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(f \times \mathsf{id}) \cdot \omega \times \mathcal{P}(f \times \mathsf{id}) \cdot \omega)\\
&= \quad \{ \alpha_{[f]} \text{ definition } \}\\
&\quad \mathcal{P}(\mathsf{id} \times ([f] \cdot \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{[f]} \times \alpha_{[f]})
\end{aligned}
$$

It remains to be investigated under what conditions step $\star$ is valid, which amounts to establish the equality

$$\mathcal{P}(f \times \mathsf{id}) \cdot \mathsf{sel} \cdot \delta_r \; = \; \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(f \times \mathsf{id}) \times \mathcal{P}(f \times \mathsf{id}))$$

Going pointwise to unfold both sides of the equation, we get

$$
\begin{aligned}
&\mathcal{P}(f \times \mathsf{id}) \cdot \mathsf{sel} \cdot \delta_r \; \langle c_1, c_2 \rangle\\
= &\{ \langle f\ (a'\theta a), \langle p_1, p_2 \rangle \rangle |\ \langle a, p_1 \rangle \in c_1 \ \wedge\ \langle a', p_2 \rangle \in c_2 \ \wedge\ a'\theta a \neq 0 \}\\
&\mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(f \times \mathsf{id}) \times \mathcal{P}(f \times \mathsf{id}))\ \langle c_1, c_2 \rangle\\
= &\{ \langle f\ a'\,\theta f\ a, \langle p_1, p_2 \rangle \rangle |\ \langle a, p_1 \rangle \in c_1 \ \wedge\ \langle a', p_2 \rangle \in c_2 \ \wedge\ f\ a'\,\theta f\ a' \neq 0 \}
\end{aligned}
$$

Therefore, both sets become equal iff

$$\forall_{a \in \mathcal{P}\pi_1\ c1, a' \in \mathcal{P}\pi_1\ c2}\ .\ a'\theta a \neq 0 \; \equiv \; fa'\,\theta f a \neq 0$$

which, expressed as a predicate on $\nu \times \nu$, reads

$$\phi\ \langle p, q \rangle \; = \; \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a'\theta a \neq 0 \; \equiv \; fa'\,\theta f a \neq 0$$

We may then conclude by §4.26, lifting ϕ to the predicate

$$\mathsf{non\_collapse\_renaming} \; = \; \Box_{\alpha}\ \phi$$

where α is the 'common' coalgebra $\alpha = \mathsf{sel} \cdot \delta_r \cdot (\alpha_{[f]} \cdot \alpha_{[f]})$.

$\square$

2. Proofs for Chapter 5

LEMMA §5.4

Proof. To complete the proof we have to show that, for arbitrary components $p : I \longrightarrow K$, $q : K \longrightarrow L$ and $r : L \longrightarrow O$,

$$\mathsf{B}(\mathsf{l} \times \mathsf{id})^I \cdot \overline{a}_{p;\mathsf{copy}_K} \;=\; \overline{a}_p \cdot \mathsf{l} \tag{D.1}$$

and

$$\mathsf{B}(\mathsf{a} \times \mathsf{id})^I \cdot \overline{a}_{(p;q);r} \;=\; \overline{a}_{p;(q;r)} \cdot \mathsf{a} \tag{D.2}$$

hold. Equation (D.1) is checked as follows:

$$\begin{array}{cl}
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot a_{p;\mathsf{copy}_K} \\
= & \quad \{\ ;\ \text{definition}\ \} \\
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \eta) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \eta \text{ strong (C.20)}\ \} \\
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\eta \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \mu \text{ and } \eta \text{ natural (C.16) (C.17)}\ \} \\
& \mu \cdot \eta \cdot \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \mu \text{ associativity (C.15) and } \mathsf{a} \text{ isomorphism}\ \} \\
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \text{routine: } \mathsf{l} = (\mathsf{l} \times \mathsf{id}) \cdot \mathsf{xr}\ \} \\
& \mathsf{Bl} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \tau_l \text{ unit (C.1)}\ \} \\
& \mathsf{l} \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \mathsf{l} \text{ natural}\ \} \\
& a_p \cdot \mathsf{l} \cdot \mathsf{xr} \\
= & \quad \{\ \text{routine: } \mathsf{l} \cdot \mathsf{xr} = \mathsf{l} \times \mathsf{id}\ \} \\
& a_p \cdot (\mathsf{l} \times \mathsf{id})
\end{array}$$

To prove (D.2) consider the following partition of the comorphism condition diagram (with $a_{(p;q);r} = \phi_1 \cdot \phi_2 \cdot \phi_3$ and $a_{p;(q;r)} = \psi_1 \cdot \psi_2 \cdot \psi_3$). Notice that each of the inner rectangles corresponds to action execution in each component p, q and r.

$$\begin{tikzcd}
((U_p \times U_q) \times U_r) \times I \arrow[r, "\mathsf{a} \times \mathsf{id}"] \arrow[d, "\phi_p"'] \arrow[ddd, bend right=60, "a_{(p;q);r}"'] & (U_p \times (U_q \times U_r) \times I \arrow[d, "\psi_p"] \arrow[ddd, bend left=60, "a_{p;(q;r)}"] \\
\mathsf{B}(U_p \times (U_q \times K)) \times U_r \arrow[r, "\mathsf{Ba} \cdot \tau_r"] \arrow[d, "\phi_q"'] & \mathsf{B}(U_p \times ((U_q \times U_r) \times K)) \arrow[d, "\psi_q"] \\
\mathsf{B}((U_p \times U_q) \times (U_r \times L)) \arrow[d, "\phi_r"'] & \mathsf{B}(U_p \times \mathsf{B}(U_q \times (U_r \times L))) \arrow[l, "\mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{B}\tau_l"'] \arrow[d, "\psi_r"] \\
\mathsf{B}(((U_p \times U_q) \times U_r) \times O) \arrow[r, "\mathsf{B}(\mathsf{a} \times \mathsf{id})"] & \mathsf{B}((U_p \times (U_q \times U_r)) \times O)
\end{tikzcd}$$

where

$$\begin{aligned}
\phi_p &= (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
\psi_p &= \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
\phi_q &= \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{BBa}^\circ \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
\psi_q &= \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_q \times \mathsf{id})) \\
\phi_r &= \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_r) \\
\psi_r &= \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mu) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{BBa}^\circ) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}\tau_l) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r))
\end{aligned}$$

Let us establish the commutativity of each of the three rectangles which jointly entail the commutativity of the overall diagram. For the upper most rectangle consider,

$$\begin{aligned}
& \mathsf{Ba} \cdot \tau_r \cdot \phi_p \\
= \quad & \{ \ \phi_p \text{ definition } \} \\
& \mathsf{Ba} \cdot \tau_r \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \tau_r \text{ natural (C.5) } \} \\
& \mathsf{Ba} \cdot \mathsf{B}((\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \text{routine: } \mathsf{a} \cdot ((\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) = (\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot \mathsf{a} \ \} \\
& \mathsf{B}((\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \tau_r \text{ associative (C.8) } \} \\
& \mathsf{B}((\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot \mathsf{a} \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mathsf{a} \text{ natural } \} \\
& \mathsf{B}((\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{a} \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \text{routine: } \mathsf{a} \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} = \mathsf{xr} \cdot (\mathsf{a} \times \mathsf{id}) \ \}
\end{aligned}$$

$$\begin{array}{cl}
 & \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \cdot (\mathsf{a} \times \mathsf{id}) \\
= & \qquad \{\ \psi_p \text{ definition } \} \\
 & \psi_p \cdot (\mathsf{a} \times \mathsf{id})
\end{array}$$

Next compute,

$$\begin{array}{cl}
 & \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{B}\tau_l \cdot \psi_q \cdot \mathsf{Ba} \cdot \tau_r \\
= & \qquad \{\ \psi_q \text{ definition } \} \\
 & \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_q \times \mathsf{id})) \cdot \mathsf{Ba} \cdot \tau_r \\
= & \qquad \{\ \mathsf{a} \text{ natural, } \tau_r \text{ natural (C.5) } \} \\
 & \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \tau_l \text{ natural (C.6) } \} \\
 & \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{BB}(\mathsf{id} \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \text{law (C.46) } \} \\
 & \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{BB}(\mathsf{id} \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{BBa} \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \tau_r \text{ natural (C.5) } \} \\
 & \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{BB}(\mathsf{id} \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{BBa} \cdot \mathsf{B}\tau_r \cdot \tau_r \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \mu \text{ natural (C.16) } \} \\
 & \mathsf{Ba}^\circ \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{Ba} \cdot \mu \cdot \mathsf{B}\tau_r \cdot \tau_r \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \text{routine: } \mathsf{a} \cdot \mathsf{xr} \cdot (\mathsf{a}^\circ \times \mathsf{id}) = \mathsf{a}^\circ \cdot (\mathsf{id} \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{a} \ \} \\
 & \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mu \cdot \mathsf{B}\tau_r \cdot \tau_r \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \mu \text{ strong (C.19) } \} \\
 & \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \tau_r \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \tau_r \text{ natural (C.5) } \} \\
 & \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Ba}^\circ \times \mathsf{id}) \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \mu \text{ natural (C.16) } \} \\
 & \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{BBa}^\circ \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_q) \times \mathsf{id}) \\
= & \qquad \{\ \phi_q \text{ definition } \} \\
 & \phi_q
\end{array}$$

And, finally,

$$\begin{array}{cl}
 & \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \phi_r \cdot \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{B}\tau_l \\
= & \qquad \{\ \phi_r \text{ definition } \}
\end{array}$$

$$
\begin{array}{ll}
 & \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_r) \cdot \mathsf{Ba}^{\circ} \cdot \mu \cdot \mathsf{B}\tau_l \\
= & \qquad \{\ \mathsf{a}^{\circ}, \tau_l, \mu \text{ natural } \} \\
 & \mu \cdot \mathsf{BB}(\mathsf{a} \times \mathsf{id}) \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{Ba}^{\circ} \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r)) \\
= & \qquad \{\ \text{routine: } (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{a}^{\circ} = \mathsf{a}^{\circ} \cdot (\mathsf{id} \times \mathsf{a}^{\circ}) \cdot \mathsf{a} \ \} \\
 & \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{a}^{\circ}) \cdot \mathsf{BBa} \cdot \mathsf{B}\tau_l \cdot \mathsf{Ba}^{\circ} \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r)) \\
= & \qquad \{\ \tau_l \text{ associative (C.4) } \} \\
 & \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{a}^{\circ}) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mathsf{Ba} \cdot \mathsf{Ba}^{\circ} \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r)) \\
= & \qquad \{\ \tau_l \text{ natural, } \mathsf{a} \text{ isomorphism } \} \\
 & \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Ba}^{\circ}) \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r)) \\
= & \qquad \{\ \mu \text{ natural (C.16) } \} \\
 & \mu \cdot \mathsf{BBa}^{\circ} \cdot \mu \cdot \mathsf{BB}\tau_l \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{Ba}^{\circ}) \cdot \mathsf{BB}(\mathsf{id} \times \tau_l) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r)) \\
= & \qquad \{\ \tau_l \text{ natural (C.6) } \} \\
 & \mu \cdot \mathsf{BBa}^{\circ} \cdot \mu \cdot \mathsf{BB}\tau_l \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{BBa}^{\circ}) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}\tau_l) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r)) \\
= & \qquad \{\ \mu \text{ strong (C.21) } \} \\
 & \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mu) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{BBa}^{\circ}) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}\tau_l) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_r)) \\
= & \qquad \{\ \psi_q \text{ definition } \} \\
 & \psi_r
\end{array}
$$

□

$\boxed{\text{LEMMA §5.20}}$

Proof. We shall successively verify that

$$p[f,\mathsf{id}] \sim \ulcorner f \urcorner \,;\, p \tag{D.3}$$

$$p[\mathsf{id},g] \sim p \,;\, \ulcorner g \urcorner \tag{D.4}$$

to conclude, by law (5.9) in §5.19, that

$$p[f,g] = (p[f,\mathsf{id}])[\mathsf{id},g] \sim (\ulcorner f \urcorner \,;\, p)[\mathsf{id},g] \sim \ulcorner f \urcorner \,;\, p \,;\, \ulcorner g \urcorner$$

Therefore, we first prove that r is a Cp arrow from $\ulcorner f \urcorner \,;\, p$ to $p[f,\mathsf{id}]$. As seeds are trivially preserved, this establishes (D.3). A similar argument, in terms of l, proves (D.4). Thus,

$$\begin{array}{ll}
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot a_{\ulcorner f \urcorner ; p} \\
= & \quad \{ \text{ ; and \textit{function lifting} definitions } \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\eta \times \mathsf{id}) \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \eta \text{ strong (C.18) } \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \eta \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \eta \text{ natural (C.17) } \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mu \cdot \eta \cdot \mathsf{Ba}^{\circ} \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ monad unit (C.14) } \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{Ba}^{\circ} \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ routine: } (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{a}^{\circ} = \mathsf{r} \ \} \\
 & \mathsf{Br} \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \tau_l \text{ unit (C.2) } \} \\
 & \mathsf{r} \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot ((\mathsf{id} \times f) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \mathsf{xr}, \mathsf{a} \text{ natural and } \mathsf{xr} = \mathsf{xr}^{\circ} \ \} \\
 & \mathsf{r} \cdot (\mathsf{id} \times a_p) \cdot (\mathsf{id} \times (\mathsf{id} \times f)) \cdot \mathsf{a} \\
= & \quad \{ \ \mathsf{r} \text{ natural } \} \\
 & a_p \cdot (\mathsf{id} \times f) \cdot \mathsf{r} \cdot \mathsf{a} \\
= & \quad \{ \text{ routine: } \mathsf{r} \cdot \mathsf{a} = (\mathsf{r} \times \mathsf{id}) \ \} \\
 & a_p \cdot (\mathsf{id} \times f) \cdot (\mathsf{r} \times \mathsf{id}) \\
= & \quad \{ \text{ \textit{wrapping} definition } \} \\
 & a_{p[f,\mathsf{id}]} \cdot (\mathsf{r} \times \mathsf{id})
\end{array}$$

Finally, to establish (D.4),

$$\begin{array}{ll}
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot a_{p;\ulcorner g \urcorner} \\
= & \qquad \{ \ \mathbf{;} \text{ and } \textit{function lifting} \text{ definitions } \} \\
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \eta) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times g)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \eta \text{ strong and natural (C.20) and (C.17) } \} \\
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mu \cdot \mathsf{B}\eta \cdot \mathsf{Ba}^{\circ} \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times g)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \text{monad unit (C.14) } \} \\
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{Ba}^{\circ} \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times g)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \mathsf{a} \text{ natural isomorphism } \} \\
& \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \text{functors } \} \\
& \mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \text{routine: } (\mathsf{l} \times \mathsf{id}) \cdot \mathsf{xr} = \mathsf{l} \ \} \\
& \mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{Bl} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \tau_r \text{ unit (C.1) } \} \\
& \mathsf{B}(\mathsf{id} \times g) \cdot \mathsf{l} \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \mathsf{l} \text{ natural } \} \\
& \mathsf{B}(\mathsf{id} \times g) \cdot a_p \cdot \mathsf{l} \cdot \mathsf{xr} \\
= & \qquad \{ \ \text{routine: } \mathsf{l} \cdot \mathsf{xr} = \mathsf{l} \times \mathsf{id} \ \} \\
& \mathsf{B}(\mathsf{id} \times g) \cdot a_p \cdot (\mathsf{l} \times \mathsf{id}) \\
= & \qquad \{ \ \textit{wrapping} \text{ definition } \} \\
& a_{p[\mathsf{id},g]} \cdot (\mathsf{l} \times \mathsf{id})
\end{array}$$

□

$\boxed{\text{LEMMA §5.24}}$

Proof. To complete the proof of Lemma §5.24, we first show that $h \times k$ is a comorphism from (coalgebra underlying) $p \boxplus q$ to $p' \boxplus q'$:

$$
\begin{array}{cl}
 & a_{p' \boxplus q'} \cdot ((h \times k) \times \mathsf{id}) \\
= & \quad \{ \ \boxplus \text{ definition } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'}) \\
 & \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \cdot ((h \times k) \times \mathsf{id}) \\
= & \quad \{ \ \mathsf{dr}, \mathsf{xr} \text{ and } \mathsf{a} \text{ natural } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'}) \\
 & \cdot (h \times \mathsf{id}) \times k + h \times (k \times \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \text{assumption: } h : p \longrightarrow p' \text{ and } k : q \longrightarrow q' \ \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot \mathsf{B}(h \times \mathsf{id}) \times k + h \times \mathsf{B}(k \times \mathsf{id}) \\
 & \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \tau_r, \tau_l \text{ natural (C.5) and (C.6) } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot \mathsf{B}((h \times \mathsf{id}) \times k) + \mathsf{B}(h \times (k \times \mathsf{id})) \\
 & \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \mathsf{xr}, \mathsf{a}^\circ \text{ natural } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot \mathsf{B}((h \times k) \times \mathsf{id}) + \mathsf{B}((h \times k) \times \mathsf{id}) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \\
 & \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ absorption } \} \\
 & [\mathsf{B}((h \times k) \times \iota_1), \mathsf{B}((h \times k) \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
 & \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \text{functors, + fusion } \} \\
 & \mathsf{B}((h \times k) \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \\
 & \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \boxplus \text{ definition } \} \\
 & \mathsf{B}((h \times k) \times \mathsf{id}) \cdot a_{p \boxplus q}
\end{array}
$$

The next proof obligation aims to establish r_1° as a comorphism from (the underlying coalgebra of) $\mathsf{copy}_{K \boxplus K'}$ to $\mathsf{copy}_K \boxplus \mathsf{copy}_{K'}$:

$$
a_{\mathsf{copy}_{K \boxplus K'}} \cdot (\mathsf{r}^\circ \times \mathsf{id})
$$

$$
\begin{array}{ll}
= & \quad \{ \ \boxplus \text{ definition} \ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{Bxr}+\mathsf{Ba}^\circ)\cdot(\tau_r+\tau_l)\cdot(a_{\mathsf{copy}_K}\times\mathsf{id}+\mathsf{id}\times a_{\mathsf{copy}_{K'}}) \\
 & \cdot(\mathsf{xr}+\mathsf{a})\cdot\mathsf{dr}\cdot(\mathsf{r}^\circ\times\mathsf{id}) \\
= & \quad \{ \ \mathsf{copy} \text{ definition} \ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{Bxr}+\mathsf{Ba}^\circ)\cdot(\tau_r+\tau_l)\cdot(\eta\times\mathsf{id}+\mathsf{id}\times\eta)\cdot(\mathsf{xr}+\mathsf{a}) \\
 & \cdot\mathsf{dr}\cdot(\mathsf{r}^\circ\times\mathsf{id}) \\
= & \quad \{ \ \eta \text{ strong (C.18) and (C.20)} \ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{Bxr}+\mathsf{Ba}^\circ)\cdot(\eta+\eta)\cdot(\mathsf{xr}+\mathsf{a})\cdot\mathsf{dr}\cdot(\mathsf{r}^\circ\times\mathsf{id}) \\
= & \quad \{ \ \eta \text{ (C.17), } \mathsf{dr} \text{ natural} \ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{Bxr}+\mathsf{Ba}^\circ)\cdot(\mathsf{Bxr}+\mathsf{Ba})\cdot(\mathsf{B}(\mathsf{r}^\circ\times\mathsf{id})+\mathsf{B}(\mathsf{r}^\circ\times\mathsf{id})) \\
 & \cdot(\eta+\eta)\cdot\mathsf{dr} \\
= & \quad \{ \ \mathsf{xr}, \mathsf{a} \text{ isomorphisms} \ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{B}(\mathsf{r}^\circ\times\mathsf{id})+\mathsf{B}(\mathsf{r}^\circ\times\mathsf{id}))(\eta+\eta)\cdot\mathsf{dr} \\
= & \quad \{ \ + \text{ absorption} \ \} \\
 & [\mathsf{B}(\mathsf{r}^\circ\times\iota_1),\mathsf{B}(\mathsf{r}^\circ\times\iota_2)]\cdot(\eta+\eta)\cdot\mathsf{dr} \\
= & \quad \{ \ + \text{ fusion} \ \} \\
 & \mathsf{B}(\mathsf{r}^\circ\times\mathsf{id})\cdot[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\eta+\eta)\cdot\mathsf{dr} \\
= & \quad \{ \ \text{law (C.64)} \ \} \\
 & \mathsf{B}(\mathsf{r}^\circ\times\mathsf{id})\cdot\eta\cdot\mathsf{dr}^\circ\cdot\mathsf{dr} \\
= & \quad \{ \ \mathsf{dr} \text{ isomorphism} \ \} \\
 & \mathsf{B}(\mathsf{r}^\circ\times\mathsf{id})\cdot\eta \\
= & \quad \{ \ \mathsf{copy} \text{ definition} \ \} \\
 & \mathsf{B}(\mathsf{r}^\circ\times\mathsf{id})\cdot a_{\mathsf{copy}_{K\boxplus K'}}
\end{array}
$$

Finally, we show that $\mathsf{m} : U \longrightarrow V$, where $U = (U_p \times U_q) \times (U_{p'} \times U_{q'})$ and $V = (U_p \times U_{p'}) \times (U_q \times U_{q'})$, is a comorphism relating the coalgebras underlying $(p\,;q) \boxplus (p'\,;q')$ and $(p \boxplus p')\ ;(q \boxplus q')$. Let us abbreviate $a_{(p;q)\boxplus(p';q')}$ and $a_{(p\boxplus p');(q\boxplus q')}$ by $a_{;\boxplus;}$ and $a_{\boxplus;\boxplus}$, respectively.
First notice that

$$a_{;\boxplus;} = [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(h_1+h_2)\cdot\mathsf{dr}$$

where

$$h_1 \ = \mathsf{Bxr}\cdot\tau_r\cdot(\mu\times\mathsf{id})\cdot(\mathsf{BBa}^\circ\times\mathsf{id})\cdot(\mathsf{B}\tau_l\times\mathsf{id})\cdot(\mathsf{B}(\mathsf{id}\times a_{p'})\times\mathsf{id})\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})\times\mathsf{id})$$

$$
\begin{aligned}
&\cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
h_2 \;&= \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times \mu) \cdot (\mathsf{id} \times \mathsf{BBa}^\circ) \cdot (\mathsf{id} \times \mathsf{B}\tau_l) \cdot (\mathsf{id} \times \mathsf{B}(\mathsf{id} \times a_{q'})) \cdot (\mathsf{id} \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \\
&\cdot (\mathsf{id} \times \tau_r) \cdot (\mathsf{id} \times (a_q \times \mathsf{id})) \cdot (\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{a}
\end{aligned}
$$

On the other hand,

$$
\begin{aligned}
a_{\boxplus;\boxplus} \;&= \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ)) \\
&\cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \\
&\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \\
&\cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}
$$

which we rewrite as a sum of two functions h_3 and h_4, such that the commutativity of the diagram below, corresponding to the comorphism condition, can be established by a number of smaller steps:

$$\begin{tikzcd}
V \times (I + I') \arrow[rr, "\mathsf{m} \times \mathsf{id}"] \arrow[d, "\mathsf{dr}"] \arrow[ddd, bend right=60, "a_{;\boxplus;}"'] & & U \times (I + I') \arrow[dl, "\mathsf{dr}"'] \arrow[ddd, "a_{\boxplus;\boxplus}"] \\
V \times I + V \times I' \arrow[r, "\mathsf{m} \times \mathsf{id} + \mathsf{m} \times \mathsf{id}"] \arrow[d, "h_1 + h_2"] & U \times I + U \times I' \arrow[d, "h_3 + h_4"] & \\
\mathsf{B}(V \times O) + \mathsf{B}(V \times O') \arrow[r, "\mathsf{B}(\mathsf{m} \times \mathsf{id}) + \mathsf{B}(\mathsf{m} \times \mathsf{id})"'] \arrow[d, "{[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]}"] & \mathsf{B}(U \times O) + \mathsf{B}(U \times O') \arrow[dr, "{[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]}"'] & \\
\mathsf{B}(V \times (O + O')) \arrow[rr, "\mathsf{B}(\mathsf{m} \times \mathsf{id})"'] & & \mathsf{B}(U \times (O + O'))
\end{tikzcd}$$

where

$$
\begin{aligned}
h_3 \;&= \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \\
&\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bxr} \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
h_4 \;&= \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Ba}^\circ) \cdot \mathsf{B}(\mathsf{id} \times \tau_l) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{a}) \\
&\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Ba}^\circ \times \mathsf{id}) \cdot (\tau_l \times \mathsf{id}) \cdot ((\mathsf{id} \times a_q) \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}
$$

The left sub-diagram commutes by definition whereas verification of the commutativity of the upper and bottom ones is a routine task. Therefore, it remains to be proved the equations corresponding to the central and rightmost sub-diagrams, *i.e.*,

$$
\begin{aligned}
a_{\boxplus;\boxplus} &= [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (h_3 + h_4) \cdot \mathsf{dr} && \text{(D.5)} \\
\mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot h_1 &= h_3 \cdot (\mathsf{m} \times \mathsf{id}) && \text{(D.6)} \\
\mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot h_2 &= h_4 \cdot (\mathsf{m} \times \mathsf{id}) && \text{(D.7)}
\end{aligned}
$$

Consider, first, equation (D.5).

$$
\begin{array}{ll}
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(h_3+h_4)\cdot\mathsf{dr} \\
= & \qquad \{\ h_3, h_4 \text{ definition}\ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mu+\mu)\cdot(\mathsf{BBa}^\circ+\mathsf{BBa}^\circ)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l)) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'}))) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot((\mathsf{Bxr}\times\mathsf{id})+(\mathsf{Ba}^\circ\times\mathsf{id}))\cdot((\tau_r\times\mathsf{id})+(\tau_l\times\mathsf{id})) \\
 & \cdot(((a_p\times\mathsf{id})\times\mathsf{id})+((\mathsf{id}\times a_q)\times\mathsf{id}))\cdot((\mathsf{xr}\times\mathsf{id})+(\mathsf{a}\times\mathsf{id}))\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ \text{routine: } (\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}=\mathsf{dl}\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}\ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mu+\mu)\cdot(\mathsf{BBa}^\circ+\mathsf{BBa}^\circ)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l)) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'}))) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot((\mathsf{Bxr}\times\mathsf{id})+(\mathsf{Ba}^\circ\times\mathsf{id}))\cdot((\tau_r\times\mathsf{id})+(\tau_l\times\mathsf{id})) \\
 & \cdot(((a_p\times\mathsf{id})\times\mathsf{id})+((\mathsf{id}\times a_q)\times\mathsf{id}))\cdot((\mathsf{xr}\times\mathsf{id})+(\mathsf{a}\times\mathsf{id}))\cdot\mathsf{dl}\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr} \\
= & \qquad \{\ \mathsf{dl} \text{ natural}\ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mu+\mu)\cdot(\mathsf{BBa}^\circ+\mathsf{BBa}^\circ)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l)) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'}))) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot\mathsf{dl}\cdot((\mathsf{Bxr}+\mathsf{Ba}^\circ)\times\mathsf{id})\cdot((\tau_r+\tau_l)\times\mathsf{id})\cdot((a_p\times\mathsf{id}+\mathsf{id}\times a_q)\times\mathsf{id}) \\
 & \cdot((\mathsf{xr}+\mathsf{a})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr} \\
= & \qquad \{\ + \text{ absorption}\ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1)\cdot\mu\cdot\mathsf{BBa}^\circ\cdot\mathsf{B}\tau_l,\mathsf{B}(\mathsf{id}\times\iota_2)\cdot\mu\cdot\mathsf{BBa}^\circ\cdot\mathsf{B}\tau_l] \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l)) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'}))) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot\mathsf{dl}\cdot((\mathsf{Bxr}+\mathsf{Ba}^\circ)\times\mathsf{id})\cdot((\tau_r+\tau_l)\times\mathsf{id})\cdot((a_p\times\mathsf{id}+\mathsf{id}\times a_q)\times\mathsf{id}) \\
 & \cdot((\mathsf{xr}+\mathsf{a})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr} \\
= & \qquad \{\ \mu \text{ natural (C.16), } \mathsf{a}^\circ \text{ natural}\ \} \\
 & [\mu\cdot\mathsf{BBa}^\circ\cdot\mathsf{BB}(\mathsf{id}\times(\mathsf{id}\times\iota_1))\cdot\mathsf{B}\tau_l,\mu\cdot\mathsf{BBa}^\circ\cdot\mathsf{BB}(\mathsf{id}\times(\mathsf{id}\times\iota_2))\cdot\mathsf{B}\tau_l]
\end{array}
$$

$$\begin{aligned}
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l))\\
&\cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'})))\\
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\\
&\cdot\mathsf{dl}\cdot((\mathsf{Bxr}+\mathsf{Ba}^\circ)\times\mathsf{id})\cdot((\tau_r+\tau_l)\times\mathsf{id})\cdot((a_p\times\mathsf{id}+\mathsf{id}\times a_q)\times\mathsf{id})\\
&\cdot((\mathsf{xr}+\mathsf{a})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}\\
=\quad&\quad\{\ +\text{ fusion}\ \}\\
&\mu\cdot\mathsf{BBa}^\circ\cdot[\mathsf{BB}(\mathsf{id}\times(\mathsf{id}\times\iota_1))\cdot\mathsf{B}\tau_l,\mathsf{BB}(\mathsf{id}\times(\mathsf{id}\times\iota_2))\cdot\mathsf{B}\tau_l]\\
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l))\\
&\cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'})))\\
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\\
&\cdot\mathsf{dl}\cdot((\mathsf{Bxr}+\mathsf{Ba}^\circ)\times\mathsf{id})\cdot((\tau_r+\tau_l)\times\mathsf{id})\cdot((a_p\times\mathsf{id}+\mathsf{id}\times a_q)\times\mathsf{id})\\
&\cdot((\mathsf{xr}+\mathsf{a})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}\\
=\quad&\quad\{\ \tau_l\text{ natural (C.6)}\ \}\\
&\mu\cdot\mathsf{BBa}^\circ\cdot[\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times\mathsf{B}(\mathsf{id}\times\iota_1)),\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times\mathsf{B}(\mathsf{id}\times\iota_2))]\\
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l))\\
&\cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'})))\\
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\\
&\cdot\mathsf{dl}\cdot((\mathsf{Bxr}+\mathsf{Ba}^\circ)\times\mathsf{id})\cdot((\tau_r+\tau_l)\times\mathsf{id})\cdot((a_p\times\mathsf{id}+\mathsf{id}\times a_q)\times\mathsf{id})\\
&\cdot((\mathsf{xr}+\mathsf{a})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}\\
=\quad&\quad\{\ +\text{ fusion}\ \}\\
&\mu\cdot\mathsf{BBa}^\circ\cdot\mathsf{B}\tau_l\cdot[\mathsf{B}(\mathsf{id}\times\mathsf{B}(\mathsf{id}\times\iota_1)),\mathsf{B}(\mathsf{id}\times\mathsf{B}(\mathsf{id}\times\iota_2))]\\
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{Bxr})+\mathsf{B}(\mathsf{id}\times\mathsf{Ba}^\circ))\cdot(\mathsf{B}(\mathsf{id}\times\tau_r)+\mathsf{B}(\mathsf{id}\times\tau_l))\\
&\cdot(\mathsf{B}(\mathsf{id}\times(a_{p'}\times\mathsf{id}))+\mathsf{B}(\mathsf{id}\times(\mathsf{id}\times a_{q'})))\\
&\cdot(\mathsf{B}(\mathsf{id}\times\mathsf{xr})+\mathsf{B}(\mathsf{id}\times\mathsf{a}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\\
&\cdot\mathsf{dl}\cdot((\mathsf{Bxr}+\mathsf{Ba}^\circ)\times\mathsf{id})\cdot((\tau_r+\tau_l)\times\mathsf{id})\cdot((a_p\times\mathsf{id}+\mathsf{id}\times a_q)\times\mathsf{id})\\
&\cdot((\mathsf{xr}+\mathsf{a})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}\\
=\quad&\quad\{\ +\text{ absorption, functors}\ \}\\
&\mu\cdot\mathsf{BBa}^\circ\cdot\mathsf{B}\tau_l\\
&\cdot[\mathsf{B}(\mathsf{id}\times(\mathsf{B}(\mathsf{id}\times\iota_1)\cdot\mathsf{Bxr}\cdot\tau_r\cdot(a_{p'}\times\mathsf{id})\cdot\mathsf{xr})),\mathsf{B}(\mathsf{id}\times(\mathsf{B}(\mathsf{id}\times\iota_1)\cdot\mathsf{Ba}^\circ\cdot\tau_l\cdot(\mathsf{id}\times a_{q'})\cdot\mathsf{a}))]\\
&\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\cdot\mathsf{dl}\cdot((\mathsf{Bxr}+\mathsf{Ba}^\circ)\times\mathsf{id})\cdot((\tau_r+\tau_l)\times\mathsf{id})\\
&\cdot((a_p\times\mathsf{id}+\mathsf{id}\times a_q)\times\mathsf{id})\cdot((\mathsf{xr}+\mathsf{a})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}\\
=\quad&\quad\{\ +\text{ fusion, }+\text{ cancellation, }+\text{ absorption}\ \}\\
&\mu\cdot\mathsf{BBa}^\circ\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[(\mathsf{id}\times\iota_1),(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(\mathsf{Bxr}+\mathsf{Ba}^\circ))
\end{aligned}$$

$$
\begin{aligned}
& \cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \\
& \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mathsf{dr} \text{ isomorphism} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ)) \\
& \cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}^\circ) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \\
& \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \\
& \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mathsf{dr}^\circ \text{ definition} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ)) \\
& \cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{id} \times \iota_1, \mathsf{id} \times \iota_2]) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \\
& \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \\
& \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ + \text{ fusion, } + \text{ cancellation} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ)) \\
& \cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot [\mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_1)), \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_2))] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \\
& \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \\
& \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ + \text{ absorption} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ)) \\
& \cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot [\mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_1)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}), \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_2)) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})] \\
& \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \\
& \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mathsf{a} \text{ natural, } \mathsf{xr} \text{ natural} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ)) \\
& \cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot [\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id})] \\
& \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \\
& \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}
$$

$=\quad \{ + \text{ fusion} \}$

$$\begin{aligned}
&\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ))\\
&\cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a}))\\
&\cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot [\mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id})]\\
&\cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id})\\
&\cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

$=\quad \{ + \text{ absorption} \}$

$$\begin{aligned}
&\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ))\\
&\cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a}))\\
&\cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot [\mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}) \cdot \tau_r, \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id}) \cdot \tau_r]\\
&\cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id})\\
&\cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

$=\quad \{ \tau_r \text{ natural (C.5)} \}$

$$\begin{aligned}
&\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ))\\
&\cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a}))\\
&\cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot [\tau_r \cdot (\mathsf{B}(\mathsf{id} \times \iota_1) \times \mathsf{id}), \tau_r \cdot (\mathsf{B}(\mathsf{id} \times \iota_2) \times \mathsf{id})]\\
&\cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id})\\
&\cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

$=\quad \{ + \text{ fusion} \}$

$$\begin{aligned}
&\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ))\\
&\cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a}))\\
&\cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \times \mathsf{id}, \mathsf{B}(\mathsf{id} \times \iota_2) \times \mathsf{id}]\\
&\cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id})\\
&\cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

$=\quad \{ + \text{ fusion}, + \text{ cancellation} \}$

$$\begin{aligned}
&\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ))\\
&\cdot \mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a}))\\
&\cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id})\\
&[\iota_1 \times \mathsf{id}, \iota_2 \times \mathsf{id}] \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id})\\
&\cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

$=\quad \{ \mathsf{dl}^\circ \text{ definition} \}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ))$$

$$\begin{aligned}
& \cdot\mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot\mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \\
& \mathsf{dl}^\circ \cdot \mathsf{dl} \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \\
& \cdot((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mathsf{dl} \text{ isomorphism } \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ)) \\
& \cdot\mathsf{B}(\mathsf{id} \times (\tau_r + \tau_l)) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id} + \mathsf{id} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot\mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1) \times \mathsf{id}, \mathsf{B}(\mathsf{id} \times \iota_2) \times \mathsf{id}] \times \mathsf{id}) \\
& \cdot((\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}) \cdot ((\tau_r + \tau_l) \times \mathsf{id}) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id}) \\
& \cdot((\mathsf{xr} + \mathsf{a}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ a_{\boxplus;\boxplus} \text{ definition } \} \\
& a_{\boxplus;\boxplus}
\end{aligned}$$

And, finally, equation (D.6) (the proof of (D.7) follows a similar argument):

$$\begin{aligned}
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot h_1 \\
= \quad & \{ \ h_1 \text{ definition } \} \\
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (\mu \times \mathsf{id}) \cdot (\mathsf{BBa}^\circ \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id}) \\
& \cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mu \text{ strong (C.19) } \} \\
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \mu \cdot \mathsf{B}\tau_r \cdot \tau_r \cdot (\mathsf{BBa}^\circ \times \mathsf{id}) \cdot (\mathsf{B}\tau_l \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id}) \\
& \cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \tau_r \text{ natural (C.5) } \} \\
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \mu \cdot \mathsf{BB}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id}) \\
& \cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \mathsf{a} \text{ isomorphism } \} \\
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \mu \cdot \mathsf{BB}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \mathsf{Ba} \cdot \tau_r \\
& \cdot(\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{ \ \text{law (C.35) } \} \\
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \mu \cdot \mathsf{BB}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Ba} \cdot \tau_r \\
& \cdot(\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

$$= \quad \{\ \mu \text{ natural (C.16)}\ \}$$

$$\mu \cdot \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{BBxr} \cdot \mathsf{BB}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Ba} \cdot \tau_r$$
$$\cdot(\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{routine: } \mathsf{xr} \cdot (\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{a}^\circ = (\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{xr})\ \}$$

$$\mu \cdot \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{BB}(\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Ba} \cdot \tau_r$$
$$\cdot(\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id}) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{routine: } (\mathsf{m} \times \mathsf{id}) \cdot (\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{xr}) = \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{m} \cdot \mathsf{a}^\circ\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{BBm} \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id})$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{law (C.35)}\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{BBm} \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \mathsf{Ba}^\circ \cdot \mathsf{Ba} \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id})$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \mathsf{a} \text{ isomorphism}\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{BBm} \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(\tau_l \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id})$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{law (C.44)}\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Bm} \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id})$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \tau_l \text{ natural (C.6)}\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Bm} \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{id})$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \tau_r \text{ natural (C.5)}\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{Bm} \cdot \mathsf{B}((\mathsf{id} \times a_{p'}) \times \mathsf{id}) \cdot \tau_r$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \mathsf{m} \text{ natural}\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{Bm} \cdot \tau_r$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \tau_r \text{ natural (C.5)}\ \}$$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{Bm}$$
$$\cdot\mathsf{B}((\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{routine: } \mathsf{m} \cdot ((\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{id}) = (\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{m}\ \}$$

$$\mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})$$
$$\cdot \mathsf{B}(\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{Bm} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ law (C.42) $\}$

$$\mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})$$
$$\cdot \mathsf{B}(\mathsf{xr} \times \mathsf{id}) \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{m} \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ τ_r natural (C.5) $\}$

$$\mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})$$
$$\cdot \tau_r \cdot (\mathsf{Bxr} \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{m} \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ m natural $\}$

$$\mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})$$
$$\cdot \tau_r \cdot (\mathsf{Bxr} \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot \mathsf{m} \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr}$$

$= \quad \{$ routine: $\mathsf{m} \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} = (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \cdot (\mathsf{m} \times \mathsf{id})$ $\}$

$$\mu \cdot \mathsf{BBa}^{\circ} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bxr}) \cdot \mathsf{B}(\mathsf{id} \times \tau_r) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{xr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})$$
$$\cdot \tau_r \cdot (\mathsf{Bxr} \times \mathsf{id}) \cdot (\tau_r \times \mathsf{id}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id}) \cdot (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{xr} \cdot (\mathsf{m} \times \mathsf{id})$$

$= \quad \{$ h_3 definition $\}$

$$h_3 \cdot (\mathsf{m} \times \mathsf{id})$$

□

$\boxed{\text{LEMMA §5.25}}$

Proof. We prove that $\mathsf{r} : \mathbf{1} \times \mathbf{1} \longrightarrow \mathbf{1}$ is a comorphism from $\ulcorner f \urcorner \boxplus \ulcorner g \urcorner$ to $\ulcorner f + g \urcorner$.

$$\begin{aligned}
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot a_{\ulcorner f \urcorner \boxplus \ulcorner g \urcorner} \\
= \quad & \{ \ \boxplus \text{ and } \textit{lifting} \text{ definitions } \} \\
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (\eta \times \mathsf{id} + \mathsf{id} \times \eta) \\
& \cdot((\mathsf{id} \times f) \times \mathsf{id} + \mathsf{id} \times (\mathsf{id} \times g)) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= \quad & \{ \ \eta \text{ strong (C.18) and (C.20) } \} \\
& \mathsf{Br} \times \mathsf{id} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\eta + \eta) \\
& \cdot((\mathsf{id} \times f) \times \mathsf{id} + \mathsf{id} \times (\mathsf{id} \times g)) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= \quad & \{ \ \eta \text{ natural (C.17) } \} \\
& \mathsf{Br} \times \mathsf{id} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\eta + \eta) \\
& \cdot((\mathsf{id} \times f) \times \mathsf{id} + \mathsf{id} \times (\mathsf{id} \times g)) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= \quad & \{ \ \mathsf{xr}, \mathsf{a} \text{ natural } \} \\
& \mathsf{Br} \times \mathsf{id} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\eta + \eta) \cdot (\mathsf{xr} + \mathsf{a}^\circ) \\
& (\mathsf{xr} + \mathsf{a}) \cdot (\mathsf{id} \times f + \mathsf{id} \times g) \cdot \mathsf{dr} \\
= \quad & \{ \ \mathsf{xr}, \mathsf{a} \text{ isomorphisms } \} \\
& \mathsf{Br} \times \mathsf{id} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\eta + \eta) \cdot (\mathsf{id} \times f + \mathsf{id} \times g) \cdot \mathsf{dr} \\
= \quad & \{ \ \text{law (C.64) } \} \\
& \mathsf{Br} \times \mathsf{id} \cdot \eta \cdot \mathsf{dr}^\circ \cdot ((\mathsf{id} \times f) + (\mathsf{id} \times g)) \cdot \mathsf{dr} \\
= \quad & \{ \ \eta, \mathsf{dr} \text{ natural } \} \\
& \eta \cdot (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{dr}^\circ \cdot \mathsf{dr} \cdot (\mathsf{id} \times (f + g)) \\
= \quad & \{ \ \mathsf{dr} \text{ isomorphism, functors } \} \\
& \eta \cdot (\mathsf{id} \times (f + g)) \cdot (\mathsf{r} \times \mathsf{id}) \\
= \quad & \{ \ \textit{lifting} \text{ definition } \} \\
& a_{\ulcorner f + g \urcorner} \cdot (\mathsf{r} \times \mathsf{id})
\end{aligned}$$

□

LEMMA §5.26

Proof.
Associativity: We prove the comorphism condition by establishing the commutativity of the diagram below which corresponds to the following equivalent formulation of associativity:

$$((p \boxplus q) \boxplus r)[\mathsf{id}, \mathsf{a}_+] = (p \boxplus (q \boxplus r))[\mathsf{a}_+, \mathsf{id}]$$

$$\begin{tikzcd}
((U_p \times U_q) \times U_r) \times ((I + J) + K) \arrow[r, "\mathsf{a} \times \mathsf{a}_+"] \arrow[d, "a_{(p \boxplus q) \boxplus r}"'] & (U_p \times (U_q \times U_r)) \times (I + (J + K)) \arrow[d, "a_{p \boxplus (q \boxplus r)}"] \\
\mathsf{B}(((U_p \times U_q) \times U_r) \times ((O + P) + R)) \arrow[r, "\mathsf{B}(\mathsf{a} \times \mathsf{a}_+)"] & \mathsf{B}((U_p \times (U_q \times U_r)) \times (O + (P + R)))
\end{tikzcd}$$

First notice that, by functoriality, $a_{(p \boxplus q) \boxplus r}$ and $a_{p \boxplus (q \boxplus r)}$ can be written as

$$\begin{aligned}
a_{(p \boxplus q) \boxplus r} &= [\mathsf{B}(\mathsf{id} \times \iota_1), (\mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\phi_1 + \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_r)) \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id} + \mathsf{id}) \\
&\quad \cdot (\mathsf{dr} \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
a_{p \boxplus (q \boxplus r)} &= [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) + \phi_2) \cdot (\mathsf{id} + \mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
&\quad \cdot (\mathsf{id} + \mathsf{id} \times \mathsf{dr}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}
\end{aligned}$$

where

$$\begin{aligned}
\phi_1 &= \mathsf{Bxr} \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id} \cdot (\tau_r + \tau_l) \times \mathsf{id} \\
&\quad \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
\phi_2 &= \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{id} \times (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot \mathsf{id} \times (\tau_r + \tau_l) \\
&\quad \cdot \mathsf{id} \times (a_q \times \mathsf{id} + \mathsf{id} \times a_r)
\end{aligned}$$

The proof becomes easier if equivalent formulations of $a_{(p \boxplus q) \boxplus r}$ and $a_{(p \boxplus (q \boxplus r)}$ are found such that the a_p, a_q and a_r appear in the expression as independent parcels, replacing, for example,

$$\cdots (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} + (\mathsf{id} \times a_r) \cdots$$

by

$$\cdots (a_p \times \mathsf{id} + \mathsf{id} \times a_q) + (\mathsf{id} \times a_r) \cdots$$

This can be achieved by an extra application of distributivity. Therefore, we define morphisms α and β by replacing ϕ_1 and ϕ_2 in the original equations for $a_{(p \boxplus q) \boxplus r}$ and $a_{p \boxplus (q \boxplus r)}$, respectively, by ψ_1 and ψ_2 given by

$$\begin{aligned}
\psi_1 &= [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \\
&\quad \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{a} + \mathsf{xr}) \cdot \mathsf{dl} \\
\psi_2 &= [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr} \times \mathsf{id}) + \mathsf{B}(\mathsf{a} \times \mathsf{id})) \cdot (\mathsf{Ba}^\circ + \mathsf{Ba}^\circ) \cdot (\tau_l + \tau_l) \\
&\quad \cdot (\mathsf{id} \times a_q + \mathsf{id} \times a_r) \cdot (\mathsf{xr} \cdot \mathsf{a}^\circ + \mathsf{a}^\circ) \cdot \mathsf{dr}
\end{aligned}$$

We prove now that $\phi_1 = \psi_1$. A similar calculation establishes $\phi_2 = \psi_2$. Thus,

$$
\begin{array}{cl}
 & \psi_1 \\
= & \quad \{\ \mathsf{a}, \mathsf{xr} \text{ natural } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \\
 & \cdot(\mathsf{a} + \mathsf{xr}) \cdot ((a_p \times \mathsf{id}) \times \mathsf{id} + (\mathsf{id} \times a_q) \times \mathsf{id}) \cdot \mathsf{dl} \\
= & \quad \{\ \mathsf{dl} \text{ natural } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \\
 & \cdot(\mathsf{a} + \mathsf{xr}) \cdot \mathsf{dl} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ \text{laws (C.8) and (C.38) } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \\
 & \cdot(\mathsf{Ba} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) + \mathsf{Bxr} \cdot \tau_r \cdot (\tau_l \times \mathsf{id})) \cdot \mathsf{dl} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ \mathsf{dl} \text{ natural } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \\
 & \cdot(\mathsf{Ba} \cdot \tau_r + \mathsf{Bxr} \cdot \tau_r) \cdot \mathsf{dl} \cdot (\tau_r + \tau_l) \times \mathsf{id} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ \text{routine: } (\mathsf{a}^\circ \times \mathsf{id}) \cdot \mathsf{xr} \cdot \mathsf{a} = \mathsf{xr} \cdot (\mathsf{xr} \times \mathsf{id}) \text{ and } (\mathsf{xr} \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot \mathsf{xr} = \mathsf{xr} \cdot (\mathsf{a}^\circ \times \mathsf{id})\ \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Bxr}) \cdot (\mathsf{B}(\mathsf{xr} \times \mathsf{id}) + \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id})) \cdot (\tau_r + \tau_r) \\
 & \cdot\mathsf{dl} \cdot (\tau_r + \tau_l) \times \mathsf{id} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ \tau_r \text{ natural (C.5) } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Bxr}) \cdot (\tau_r + \tau_r) \cdot (\mathsf{Bxr} \times \mathsf{id} + \mathsf{Ba}^\circ \times \mathsf{id}) \\
 & \cdot\mathsf{dl} \cdot (\tau_r + \tau_l) \times \mathsf{id} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ \mathsf{dl} \text{ natural } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Bxr}) \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id} \\
 & \cdot(\tau_r + \tau_l) \times \mathsf{id} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ + \text{ absorption } \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Bxr}] \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id} \\
 & \cdot(\tau_r + \tau_l) \times \mathsf{id} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ \mathsf{xr} \text{ natural and } + \text{ fusion } \} \\
 & \mathsf{Bxr} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \times \mathsf{id}, \mathsf{B}(\mathsf{id} \times \iota_2) \times \mathsf{id}] \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id} \\
 & \cdot(\tau_r + \tau_l) \times \mathsf{id} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= & \quad \{\ \text{law (C.66) } \} \\
 & \mathsf{Bxr} \cdot \tau_r \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id} \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \times \mathsf{id}
\end{array}
$$

$$
\begin{aligned}
& \cdot(\tau_r + \tau_l) \times \mathsf{id} \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \times \mathsf{id} \\
= \quad & \{ \ \phi_1 \text{ definition } \} \\
& \phi_1
\end{aligned}
$$

We can now establish the commutativity of the diagram above by showing that

$$\beta \cdot (\mathsf{a} \times \mathsf{a}_+) \ = \ \mathsf{B}(\mathsf{a} \times \mathsf{a}_+) \cdot \alpha$$

The reasoning is as follows:

$$
\begin{aligned}
& \beta \cdot (\mathsf{a} \times \mathsf{a}_+) \\
= \quad & \{ \ \beta \text{ definition } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \\
& \cdot(\mathsf{id} + (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr} \times \mathsf{id}) + \mathsf{B}(\mathsf{a} \times \mathsf{id}))) \cdot (\mathsf{Bxr} + (\mathsf{Ba}^\circ + \mathsf{Ba}^\circ)) \cdot (\tau_r + (\tau_l + \tau_l)) \\
& \cdot(a_p \times \mathsf{id} + (\mathsf{id} \times a_q + \mathsf{id} \times a_r)) \cdot (\mathsf{id} + (\mathsf{xr} \cdot \mathsf{a}^\circ + \mathsf{a}^\circ)) \cdot (\mathsf{id} + \mathsf{dr}) \cdot (\mathsf{id} + \mathsf{id} \times (\mathsf{xr} + \mathsf{a})) \\
& \cdot(\mathsf{id} + \mathsf{id} \times \mathsf{dr}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \cdot (\mathsf{a} \times \mathsf{a}_+) \\
= \quad & \{ \ \text{routine verification } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \\
& \cdot(\mathsf{id} + (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr} \times \mathsf{id}) + \mathsf{B}(\mathsf{a} \times \mathsf{id}))) \cdot (\mathsf{Bxr} + (\mathsf{Ba}^\circ + \mathsf{Ba}^\circ)) \cdot (\tau_r + (\tau_l + \tau_l)) \\
& \cdot(a_p \times \mathsf{id} + (\mathsf{id} \times a_q + \mathsf{id} \times a_r)) \cdot (\mathsf{id} + (\mathsf{xr} \cdot \mathsf{a}^\circ + \mathsf{a}^\circ)) \cdot \mathsf{a}_+ \cdot ((\mathsf{a} + \mathsf{a}) + \mathsf{a}) \cdot (\mathsf{dl} + \mathsf{id}) \\
& \cdot((\mathsf{xr} + \mathsf{a}) \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= \quad & \{ \ \mathsf{a}_+ \text{ natural } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{a}_+ \\
& \cdot((\mathsf{id} + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr} \times \mathsf{id})) + \mathsf{B}(\mathsf{a} \times \mathsf{id})) \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) + \mathsf{Ba}^\circ) \cdot ((\tau_r + \tau_l) + \tau_l) \\
& \cdot((a_p \times \mathsf{id} + \mathsf{id} \times a_q) + \mathsf{id} \times a_r) \cdot ((\mathsf{id} + \mathsf{xr} \cdot \mathsf{a}^\circ) + \mathsf{a}^\circ) \cdot ((\mathsf{a} + \mathsf{a}) + \mathsf{a}) \cdot (\mathsf{dl} + \mathsf{id}) \cdot \\
& ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= \quad & \{ \ \mathsf{a} \text{ isomorphism } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{a}_+ \\
& \cdot((\mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr} \times \mathsf{id})) + \mathsf{B}(\mathsf{a} \times \mathsf{id})) \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) + \mathsf{Ba}^\circ) \\
& \cdot((\tau_r + \tau_l) + \tau_l) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) + \mathsf{id} \times a_r) \cdot ((\mathsf{a} + \mathsf{xr}) + \mathsf{id}) \cdot (\mathsf{dl} + \mathsf{id}) \\
& \cdot((\mathsf{xr} + \mathsf{a}) \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= \quad & \{ \ \text{functors } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{a}_+ \\
& \cdot((\mathsf{B}(\mathsf{a} \times \mathsf{id}) + \mathsf{B}(\mathsf{a} \times \mathsf{id})) + \mathsf{B}(\mathsf{a} \times \mathsf{id})) \cdot (\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) + \mathsf{id}) \\
& \cdot((\mathsf{Bxr} + \mathsf{Ba}^\circ) + \mathsf{Ba}^\circ) \cdot ((\tau_r + \tau_l) + \tau_l) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) + \mathsf{id} \times a_r) \\
& \cdot((\mathsf{a} + \mathsf{xr}) + \mathsf{id}) \cdot (\mathsf{dl} + \mathsf{id}) \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}
\end{aligned}
$$

$$
\begin{array}{ll}
= & \quad \{ \ \star \ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{a}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] + \mathsf{id}) \\
& \cdot((\mathsf{B}(\mathsf{a} \times \mathsf{id}) + \mathsf{B}(\mathsf{a} \times \mathsf{id})) + \mathsf{B}(\mathsf{a} \times \mathsf{id})) \cdot (\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) + \mathsf{id}) \\
& \cdot((\mathsf{Bxr} + \mathsf{Ba}^\circ) + \mathsf{Ba}^\circ) \cdot ((\tau_r + \tau_l) + \tau_l) \cdot ((a_p \times \mathsf{id} + \mathsf{id} \times a_q) + \mathsf{id} \times a_r) \\
& \cdot((\mathsf{a} + \mathsf{xr}) + \mathsf{id}) \cdot (\mathsf{dl} + \mathsf{id}) \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ absorption and fusion } \} \\
& \mathsf{B}(\mathsf{a} \times \mathsf{a}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_1)] \cdot ([\mathsf{B}(\mathsf{id} \times \iota_2), \mathsf{B}(\mathsf{id} \times \iota_2)] + \mathsf{id}) \\
& \cdot(\mathsf{B}(\mathsf{a}^\circ \times \mathsf{id}) + \mathsf{B}(\mathsf{xr} \times \mathsf{id})) + \mathsf{id}) \cdot ((\mathsf{Bxr} + \mathsf{Ba}^\circ) + \mathsf{Ba}^\circ) \cdot ((\tau_r + \tau_l) + \tau_l) \\
& \cdot((a_p \times \mathsf{id} + \mathsf{id} \times a_q) + \mathsf{id} \times a_r) \cdot ((\mathsf{a} + \mathsf{xr}) + \mathsf{id}) \cdot (\mathsf{dl} + \mathsf{id}) \cdot ((\mathsf{xr} + \mathsf{a}) \times \mathsf{id} + \mathsf{id}) \\
& \cdot(\mathsf{dr} \times \mathsf{id} + \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \alpha \text{ definition } \} \\
& \mathsf{B}(\mathsf{a} \times \mathsf{a}_+) \cdot \alpha
\end{array}
$$

The step marked with a $\star$ is justified by the following calculation:

$$
\begin{array}{ll}
& \mathsf{B}(\mathsf{id} \times \mathsf{a}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] + \mathsf{id}) \\
= & \quad \{ \ + \text{ absorption } \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{a}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)], \mathsf{B}(\mathsf{id} \times \iota_2)] \\
= & \quad \{ \ + \text{ fusion } \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{a}_+) \cdot [[\mathsf{B}(\mathsf{id} \times \iota_1 \cdot \iota_1), \mathsf{B}(\mathsf{id} \times \iota_1 \cdot \iota_2)], \mathsf{B}(\mathsf{id} \times \iota_2)] \\
= & \quad \{ \ \mathsf{a}_+ \text{ definition } \} \\
& \mathsf{B}(\mathsf{id} \times [\mathsf{id} + \iota_1, \iota_2 \cdot \iota_2]) \cdot [[\mathsf{B}(\mathsf{id} \times \iota_1 \cdot \iota_1), \mathsf{B}(\mathsf{id} \times \iota_1 \cdot \iota_2)], \mathsf{B}(\mathsf{id} \times \iota_2)] \\
= & \quad \{ \ + \text{ fusion, } + \text{ cancellation } \} \\
& [[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_1)], \mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_2)] \\
= & \quad \{ \ + \text{ absorption, } + \text{ cancellation } \} \\
& [[\mathsf{B}(\mathsf{id} \times \iota_1), [\mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_2)]] \cdot (\mathsf{id} + \iota_1), \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), [\mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_2)]] \cdot \iota_2 \cdot \iota_2] \\
= & \quad \{ \ + \text{ fusion } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), [\mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2 \cdot \iota_2)]] \\
& \cdot[\mathsf{id} + \iota_1, \iota_2 \cdot \iota_2] \\
= & \quad \{ \ + \text{ absorption } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot [\mathsf{id} + \iota_1, \iota_2 \cdot \iota_2] \\
= & \quad \{ \ \mathsf{a}_+ \text{ definition } \}
\end{array}
$$

$$[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{a}_+$$

Rigth Unit: We prove that $\mathsf{r} : \mathbf{1} \times U_p \longrightarrow U_p$ is a comorphism establishing the commutativity of the following diagram:

$$\begin{tikzcd}
 & U_p \times (\emptyset + I) \arrow[d, "\mathsf{id} \times \mathsf{r}_+"] \\
(\mathbf{1} \times U_p) \times (\emptyset + I) \arrow[ur, "\mathsf{r} \times \mathsf{id}"] \arrow[d, "a_{\mathsf{nil} \boxplus p}"'] & U_p \times I \arrow[d, "a_p"] \\
\mathsf{B}((\mathbf{1} \times U_p) \times (\emptyset + O)) \arrow[dr, "\mathsf{B}(\mathsf{r} \times \mathsf{id})"'] & \mathsf{B}(U_p \times O) \arrow[d, "\mathsf{B}(\mathsf{id} \times \mathsf{r}_+{}^\circ)"] \\
 & \mathsf{B}(U_p \times (\emptyset + O))
\end{tikzcd}$$

$$\begin{aligned}
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot a_{\mathsf{nil} \boxplus p} \\
= \quad & \{ \ \boxplus \text{ definition} \ \} \\
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (\eta \times \mathsf{id} + \mathsf{id} \times a_p) \\
& \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= \quad & \{ \ \text{law (C.60)} \ \} \\
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (\eta \times \mathsf{id} + \mathsf{id} \times a_p) \\
& \cdot (\mathsf{xr} + \mathsf{a}) \cdot \iota_2 \cdot (\mathsf{id} \times \mathsf{r}_+) \\
= \quad & \{ \ + \text{ absorption} \ \} \\
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot (\mathsf{id} \times \mathsf{r}_+) \\
= \quad & \{ \ \mathsf{r} \text{ isomorphism} \ \} \\
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \cdot (\mathsf{r} \times \mathsf{r}_+) \\
= \quad & \{ \ \text{routine: } \mathsf{a} \cdot (\mathsf{r}^\circ \times \mathsf{id}) = \mathsf{r}^\circ \ \} \\
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{r}^\circ \cdot (\mathsf{r} \times \mathsf{r}_+) \\
= \quad & \{ \ \mathsf{r}^\circ \text{ natural} \ \} \\
& \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot \mathsf{r}^\circ \cdot a_p \cdot (\mathsf{r} \times \mathsf{r}_+) \\
= \quad & \{ \ \text{routine: } (\mathsf{id} \times \iota_2) \cdot \mathsf{r} = (\mathsf{r} \times \iota_2)\mathsf{a}^\circ \ \} \\
& \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Br} \cdot \tau_l \cdot \mathsf{r}^\circ \cdot a_p \cdot (\mathsf{r} \times \mathsf{r}_+)
\end{aligned}$$

$$= \qquad \{\ \tau_l \text{ unit (C.2) } \}$$

$$\mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{r} \cdot \cdot \mathsf{r}^\circ \cdot a_p \cdot (\mathsf{r} \times \mathsf{r}_+)$$

$$= \qquad \{\ \mathsf{r} \text{ isomorphism and } \mathsf{r}_+{}^\circ = \iota_2\ \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{r}_+{}^\circ) \cdot a_p \cdot (\mathsf{r} \times \mathsf{r}_+)$$

$$= \qquad \{\ \textit{wrapping} \text{ definition } \}$$

$$a_{p[\mathsf{r}_+, \mathsf{r}_+{}^\circ]} \cdot (\mathsf{r} \times \mathsf{id})$$

Left Unit:
We prove that $\mathsf{l} : U_p \times \mathbf{1} \longrightarrow U_p$ is a comorphism:

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot a_{p \boxplus \mathsf{nil}}$$

$$= \qquad \{\ \boxplus \text{ definition } \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times \eta)$$
$$\cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$$= \qquad \{\ \text{law (C.59) } \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times \eta)$$
$$\cdot (\mathsf{xr} + \mathsf{a}) \cdot \iota_1 \cdot (\mathsf{id} \times \mathsf{l}_+)$$

$$= \qquad \{\ + \text{ absorption and cancellation } \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \cdot (\mathsf{id} \times \mathsf{l}_+)$$

$$= \qquad \{\ \mathsf{l} \text{ isomorphism } \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \cdot (\mathsf{l}^\circ \times \mathsf{id}) \cdot (\mathsf{l} \times \mathsf{l}_+)$$

$$= \qquad \{\ \text{routine: } \mathsf{l}^\circ = \mathsf{xr} \cdot (\mathsf{l}^\circ \times \mathsf{id})\ \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{l}^\circ \cdot (\mathsf{l} \times \mathsf{l}_+)$$

$$= \qquad \{\ \mathsf{l}^\circ \text{ natural } \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot \mathsf{l}^\circ \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+)$$

$$= \qquad \{\ \tau_r \text{ unit in variant of law (C.1) } \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \mathsf{Bl}^\circ \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+)$$

$$= \qquad \{\ \text{routine: } \mathsf{l}^\circ = \mathsf{xr} \cdot (\mathsf{l}^\circ \times \mathsf{id}) \text{ and } \mathsf{xr} \text{ isomorphism } \}$$

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{B}(\mathsf{l}^\circ \times \mathsf{id}) \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+)$$

$$= \qquad \{\ \mathsf{l} \text{ isomorphism and } \mathsf{l}^\circ = \iota_1\ \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{l}^\circ) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+) \cdot (\mathsf{l} \times \mathsf{id})$$

$=$ { *wrapping* definition }

$$a_{p[\mathsf{l}_+,\mathsf{l}^\circ]} \cdot (\mathsf{l} \times \mathsf{id})$$

Commutativity:

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_{q \boxplus p} \cdot (\mathsf{id} \times \mathsf{s}_+) \cdot (\mathsf{s} \times \mathsf{id})$$

$=$ { $\boxplus$ definition }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p)$$
$$\cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \cdot (\mathsf{s} \times \mathsf{s}_+)$$

$=$ { routine: $\mathsf{dr} \cdot (\mathsf{s} \times \mathsf{s}_+) = (\mathsf{s} \times \mathsf{id} + \mathsf{s} \times \mathsf{id}) \cdot \mathsf{s}_+ \cdot \mathsf{dr}$ }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p)$$
$$\cdot (\mathsf{xr} + \mathsf{a}) \cdot (\mathsf{s} \times \mathsf{id} + \mathsf{s} \times \mathsf{id}) \cdot \mathsf{s}_+ \cdot \mathsf{dr}$$

$=$ { routine: $(\mathsf{xr} + \mathsf{a}) \cdot (\mathsf{s} \times \mathsf{id} + \mathsf{s} \times \mathsf{id}) \cdot \mathsf{s}_+ = (\mathsf{s} + \mathsf{s}) \cdot \mathsf{s}_+ \cdot (\mathsf{xr} + \mathsf{a})$ }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p)$$
$$\cdot (\mathsf{s} + \mathsf{s}) \cdot \mathsf{s}_+ \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$=$ { s, s_+ natural }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (\mathsf{s} + \mathsf{s}) \cdot \mathsf{s}_+$$
$$\cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$=$ { τ_r, τ_l interchangeable (C.7) }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\mathsf{Bs} \cdot \tau_l + \mathsf{Bs} \cdot \tau_r) \cdot \mathsf{s}_+$$
$$\cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$=$ { s_+ natural }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\mathsf{Bs} + \mathsf{Bs}) \cdot \mathsf{s}_+ \cdot (\tau_r + \tau_l)$$
$$\cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$=$ { routine: $\mathsf{xr} \cdot \mathsf{s} = (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{a}^\circ$ and $\mathsf{a}^\circ \cdot \mathsf{s} = (\mathsf{s} \times \mathsf{id}) \cdot \mathsf{xr}$ }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{s} \times \mathsf{id}) + \mathsf{B}(\mathsf{s} \times \mathsf{id})) \cdot (\mathsf{Ba}^\circ + \mathsf{Bxr}) \cdot \mathsf{s}_+ \cdot (\tau_r + \tau_l)$$
$$\cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$=$ { s_+ natural }

$$\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{s} \times \mathsf{id}) + \mathsf{B}(\mathsf{s} \times \mathsf{id})) \cdot \mathsf{s}_+ \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l)$$
$$\cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$=$ { $+$ absorption }

$$\begin{array}{ll}
 & \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{s} \times \iota_1), \mathsf{B}(\mathsf{s} \times \iota_2)] \cdot \mathsf{s}_+ \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
 & \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ + \text{ fusion, functors } \} \\
 & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot \cdot \mathsf{s}_+ \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \\
 & \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ \text{routine: } [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)] \cdot \mathsf{s}_+ = (\mathsf{id} \times \mathsf{s}_+) \cdot [(\mathsf{id} \times \iota_1), (\mathsf{id} \times \iota_2)] \ \} \\
 & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \\
 & \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ \mathsf{s}_+ = \mathsf{s}_+{}^\circ \ \} \\
 & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
 & \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ \boxplus \text{ definition } \} \\
 & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot a_{p \boxplus q}
\end{array}$$

□

$\boxed{\text{LEMMA §5.29}}$

Proof. Replacing composition with lifted functions by wrapping, equations (5.22) and (5.23) can be rewritten as

$$p \boxplus q[\iota_1, \nabla] \ \sim\ p$$

and

$$p \boxplus q[\iota_2, \nabla] \ \sim\ q$$

Then, we show that both the first and the second projection are comorphisms from the left to the right hand side. Thus,

$$\begin{array}{ll}
& \mathsf{B}(\pi_1 \times \nabla) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
& \cdot(\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_1) \\
= & \qquad \{ \text{ law (C.55) } \} \\
& \mathsf{B}(\pi_1 \times \nabla) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
& \cdot(\mathsf{xr} + \mathsf{a}) \cdot \iota_1 \\
= & \qquad \{ \ + \text{ absorption and cancellation } \} \\
& \mathsf{B}(\pi_1 \times \nabla) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot a_p \times \mathsf{id} \cdot \mathsf{xr} \\
= & \qquad \{ \text{ routine: } \nabla \cdot \iota_1 = \mathsf{id} \} \\
& \mathsf{B}(\pi_1 \times \mathsf{id}) \cdot \mathsf{Bxr} \cdot \tau_r \cdot a_p \times \mathsf{id} \cdot \mathsf{xr} \\
= & \qquad \{ \text{ routine: } (\pi_1 \times \mathsf{id}) \cdot \mathsf{xr} = \pi_1 \ \} \\
& \mathsf{B}\pi_1 \cdot \tau_r \cdot a_p \times \mathsf{id} \cdot \mathsf{xr} \\
= & \qquad \{ \text{ law (C.12) } \} \\
& \mathsf{B}\pi_1 \cdot a_p \times \mathsf{id} \cdot \mathsf{xr} \\
= & \qquad \{ \ \times \text{ definition and cancellation } \} \\
& a_p \cdot \pi_1 \cdot \mathsf{xr} \\
= & \qquad \{ \text{ routine: } (\pi_1 \times \mathsf{id}) \cdot \mathsf{xr} = \pi_1 \text{ and } \mathsf{xr} = \mathsf{xr}^\circ \ \} \\
& a_p \cdot (\pi_1 \times \mathsf{id})
\end{array}$$

which establishes (5.22). A similar calculation proves (5.23). Notice that in both cases seeds are trivially preserved. Thus,

$$\begin{array}{ll}
& \mathsf{B}(\pi_2 \times \nabla) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
& \cdot(\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_2) \\
= & \qquad \{ \text{ law (C.56) } \} \\
& \mathsf{B}(\pi_2 \times \nabla) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q)
\end{array}$$

$$
\begin{array}{ll}
& \cdot(\mathsf{xr} + \mathsf{a}) \cdot \iota_2 \\
= & \quad \{ \text{ + absorption and cancellation } \} \\
& \mathsf{B}(\pi_2 \times \nabla) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot \mathsf{id} \times a_q \cdot \mathsf{a} \\
= & \quad \{ \text{ routine: } \nabla \cdot \iota_2 = \mathsf{id} \text{ and } (\pi_2 \times \mathsf{id}) \cdot \mathsf{a}^\circ = \pi_2 \} \\
& \mathsf{B}\pi_2 \cdot \tau_l \cdot \mathsf{id} \times a_q \cdot \mathsf{a} \\
= & \quad \{ \text{ law (C.13) } \} \\
& \pi_2 \cdot a_p \times \mathsf{id} \cdot \mathsf{a} \\
= & \quad \{ \times \text{ definition and cancellation } \} \\
& a_p \cdot \pi_2 \cdot \mathsf{a} \\
= & \quad \{ \text{ routine: } (\pi_2 \times \mathsf{id}) = \pi_2 \cdot \mathsf{a} \} \\
& a_p \cdot (\pi_2 \times \mathsf{id})
\end{array}
$$

□

LEMMA §5.35

Proof. To complete the proof, the comorphism conditions mentioned in the main text have to be shown to hold. Notice that, in each case, seed preservation is obvious. Therefore, we first show that $h \times k$ is a comorphism from $p \boxtimes q$ to $p' \boxtimes q'$:

$$\begin{aligned}
& a_{p' \boxtimes q'} \cdot ((h \times k) \times \mathsf{id}) \\
= \quad & \{\ \boxtimes \text{ definition}\ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (a_{p'} \times a_{q'}) \cdot \mathsf{m} \cdot ((h \times k) \times \mathsf{id}) \\
= \quad & \{\ \mathsf{m} \text{ natural}\ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (a_{p'} \times a_{q'}) \cdot ((h \times \mathsf{id}) \times (k \times \mathsf{id})) \cdot \mathsf{m} \\
= \quad & \{\ \text{assumption: } h : p \longrightarrow p' \text{ and } k : q \longrightarrow q'\ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (\mathsf{B}(h \times \mathsf{id}) \cdot a_p \times \mathsf{B}(k \times \mathsf{id}) \cdot a_q) \cdot \mathsf{m} \\
= \quad & \{\ \delta_l \text{ natural (C.31)}\ \} \\
& \mathsf{Bm} \cdot \mathsf{B}((h \times \mathsf{id}) \times (k \times \mathsf{id})) \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= \quad & \{\ \mathsf{m} \text{ natural}\ \} \\
& \mathsf{B}((h \times k) \times \mathsf{id}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= \quad & \{\ \boxtimes \text{ definition}\ \} \\
& \mathsf{B}((h \times k) \times \mathsf{id}) \cdot a_{p \boxtimes q}
\end{aligned}$$

The next step establishes m as a comorphism from $(p \boxtimes q) \mathbin{;} (p' \boxtimes q')$ to $(p \mathbin{;} p') \boxtimes (q \mathbin{;} q')$:

$$\begin{aligned}
& a_{(p;p') \boxtimes (q;q')} \cdot (\mathsf{m} \times \mathsf{id}) \\
= \quad & \{\ \boxtimes \text{ and } \mathbf{;} \text{ definitions}\ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (\mu \times \mu) \cdot (\mathsf{BBa}^\circ \times \mathsf{BBa}^\circ) \cdot (\mathsf{B}\tau_l \times \mathsf{B}\tau_l) \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{B}(\mathsf{id} \times a_{q'})) \\
& \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (\tau_r \times \tau_r) \cdot ((a_p \times \mathsf{id}) \times (a_q \times \mathsf{id})) \cdot (\mathsf{xr} \times \mathsf{xr}) \cdot \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) \\
= \quad & \{\ \text{routine: } (\mathsf{xr} \times \mathsf{xr}) \cdot \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) = (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}\ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (\mu \times \mu) \cdot (\mathsf{BBa}^\circ \times \mathsf{BBa}^\circ) \cdot (\mathsf{B}\tau_l \times \mathsf{B}\tau_l) \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{B}(\mathsf{id} \times a_{q'})) \\
& \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (\tau_r \times \tau_r) \cdot ((a_p \times \mathsf{id}) \times (a_q \times \mathsf{id})) \cdot \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{\ \mathsf{m} \text{ natural}\ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (\mu \times \mu) \cdot (\mathsf{BBa}^\circ \times \mathsf{BBa}^\circ) \cdot (\mathsf{B}\tau_l \times \mathsf{B}\tau_l) \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{B}(\mathsf{id} \times a_{q'})) \\
& \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (\tau_r \times \tau_r) \cdot \mathsf{m} \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr} \\
= \quad & \{\ \text{law (C.75)}\ \} \\
& \mathsf{Bm} \cdot \mu \cdot \mathsf{B}\delta_l \cdot \delta_r \cdot (\mathsf{BBa}^\circ \times \mathsf{BBa}^\circ) \cdot (\mathsf{B}\tau_l \times \mathsf{B}\tau_l) \cdot (\mathsf{B}(\mathsf{id} \times a_{p'}) \times \mathsf{B}(\mathsf{id} \times a_{q'})) \\
& \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \times \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (\tau_r \times \tau_r) \cdot \mathsf{m} \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}
\end{aligned}$$

$$= \quad \{\ \delta_r \text{ natural (C.30) } \}$$

$$\mathsf{Bm} \cdot \mu \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(\mathsf{Ba}^\circ \times \mathsf{Ba}^\circ) \cdot \mathsf{B}(\tau_l \times \tau_l) \cdot \mathsf{B}((\mathsf{id} \times a_{p'}) \times (\mathsf{id} \times a_{q'}))$$
$$\cdot \mathsf{B}((\mathsf{a} \cdot \mathsf{xr}) \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \delta_r \cdot (\tau_r \times \tau_r) \cdot \mathsf{m} \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{law (C.80) } \}$$

$$\mathsf{Bm} \cdot \mu \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(\mathsf{Ba}^\circ \times \mathsf{Ba}^\circ) \cdot \mathsf{B}(\tau_l \times \tau_l) \cdot \mathsf{B}((\mathsf{id} \times a_{p'}) \times (\mathsf{id} \times a_{q'}))$$
$$\cdot \mathsf{B}((\mathsf{a} \cdot \mathsf{xr}) \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{Bm} \cdot \tau_r \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{routine: } ((\mathsf{a} \cdot \mathsf{xr}) \times (\mathsf{a} \cdot \mathsf{xr})) \cdot \mathsf{m} = \mathsf{m} \cdot (\mathsf{id} \times \mathsf{m}) \cdot \mathsf{a} \cdot \mathsf{xr} \cdot (\mathsf{m} \times \mathsf{id}) \ \}$$

$$\mathsf{Bm} \cdot \mu \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(\mathsf{Ba}^\circ \times \mathsf{Ba}^\circ) \cdot \mathsf{B}(\tau_l \times \tau_l) \cdot \mathsf{B}((\mathsf{id} \times a_{p'}) \times (\mathsf{id} \times a_{q'})) \cdot \mathsf{Bm} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m})$$
$$\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \tau_r \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \tau_r \text{ natural (C.5) } \}$$

$$\mathsf{Bm} \cdot \mu \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(\mathsf{Ba}^\circ \times \mathsf{Ba}^\circ) \cdot \mathsf{B}(\tau_l \times \tau_l) \cdot \mathsf{B}((\mathsf{id} \times a_{p'}) \times (\mathsf{id} \times a_{q'})) \cdot \mathsf{Bm} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m})$$
$$\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \mathsf{m} \text{ natural } \}$$

$$\mathsf{Bm} \cdot \mu \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(\mathsf{Ba}^\circ \times \mathsf{Ba}^\circ) \cdot \mathsf{B}(\tau_l \times \tau_l) \cdot \mathsf{Bm} \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m})$$
$$\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \delta_l \text{ natural (C.31) } \}$$

$$\mathsf{Bm} \cdot \mu \cdot \mathsf{BB}(\mathsf{a}^\circ \times \mathsf{a}^\circ) \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(\tau_l \times \tau_l) \cdot \mathsf{Bm} \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m})$$
$$\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{law (C.81) } \}$$

$$\mathsf{Bm} \cdot \mu \cdot \mathsf{BB}(\mathsf{a}^\circ \times \mathsf{a}^\circ) \cdot \mathsf{BBm} \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \delta_l) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m})$$
$$\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \mu \text{ natural (C.16) } \}$$

$$\mu \cdot \mathsf{BB}(\mathsf{m} \cdot (\mathsf{a}^\circ \times \mathsf{a}^\circ) \cdot \mathsf{m}) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \delta_l) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})$$
$$\cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \text{routine: } \mathsf{m} \cdot (\mathsf{a}^\circ \times \mathsf{a}^\circ) \cdot \mathsf{m} = (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\mathsf{id} \times \mathsf{m}) \ \}$$

$$\mu \cdot \mathsf{BB}(\mathsf{m} \times \mathsf{id}) \cdot \mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times \mathsf{m}) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \delta_l) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m})$$
$$\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \mu \text{ and } \tau_l \text{ natural (C.16) and (C.5) } \}$$

$$\mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bm}) \cdot \mathsf{B}(\mathsf{id} \times \delta_l) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m})$$
$$\cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_r \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr}$$

$$= \quad \{\ \mathsf{B} \text{ commutative } \}$$

$$
\begin{array}{ll}
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{Bm}) \cdot \mathsf{B}(\mathsf{id} \times \delta_l) \cdot \mathsf{B}(\mathsf{id} \times (a_{p'} \times a_{q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m}) \\
& \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_l \times \mathsf{id}) \cdot ((a_p \times a_q) \times \mathsf{id}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \boxtimes \text{ and } \mathbf{;} \text{ definitions } \} \\
& \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot a_{(p;p') \boxtimes (q;q')}
\end{array}
$$

Finally, we check that r° is a comorphism from $\mathsf{copy}_{K \boxtimes K'}$ to $\mathsf{copy}_K \boxtimes \mathsf{copy}_{K'}$:

$$
\begin{array}{ll}
& a_{\mathsf{copy}_K \boxtimes \mathsf{copy}_{K'}} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= & \quad \{ \ \boxtimes \text{ and } \mathsf{copy}_K \text{ definitions } \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (\eta \times \eta) \cdot \mathsf{m} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= & \quad \{ \ \text{law (C.68)} \ \} \\
& \mathsf{Bm} \cdot \eta \cdot \mathsf{m} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= & \quad \{ \ \eta \text{ natural (C.17)} \ \} \\
& \eta \cdot \mathsf{m} \cdot \mathsf{m} \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= & \quad \{ \ \mathsf{m} = \mathsf{m}^\circ \ \} \\
& \eta \cdot (\mathsf{r}^\circ \times \mathsf{id}) \\
= & \quad \{ \ \mathsf{copy}_K \text{ definition } \} \\
& a_{\mathsf{copy}_{K \boxtimes K'}}
\end{array}
$$

□

LEMMA §5.36

Proof. We prove that $\mathsf{r} : \mathbf{1} \times \mathbf{1} \longrightarrow \mathbf{1}$ is a comorphism from $\ulcorner f \urcorner \boxtimes \ulcorner g \urcorner$ to $\ulcorner f \times g \urcorner$.

$$\begin{array}{cl}
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot a_{\ulcorner f \urcorner \boxtimes \ulcorner g \urcorner} \\
= & \qquad \{ \ \boxtimes \text{ definition and } \textit{function lifting} \ \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (\eta \times \eta) \cdot ((\mathsf{id} \times f) \times (\mathsf{id} \times g)) \cdot \mathsf{m} \\
= & \qquad \{ \ \text{law (C.68)} \ \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{Bm} \cdot \eta \cdot ((\mathsf{id} \times f) \times (\mathsf{id} \times g)) \cdot \mathsf{m} \\
= & \qquad \{ \ \eta \text{ natural (C.17)} \ \} \\
 & \eta \cdot (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{m} \cdot ((\mathsf{id} \times f) \times (\mathsf{id} \times g)) \cdot \mathsf{m} \\
= & \qquad \{ \ \mathsf{m} \text{ natural} \ \} \\
 & \eta \cdot (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{m} \cdot \mathsf{m} \cdot ((\mathsf{id} \times \mathsf{id}) \times (f \times g)) \\
= & \qquad \{ \ \mathsf{m} = \mathsf{m}^\circ \ \} \\
 & \eta \cdot (\mathsf{r} \times \mathsf{id}) \cdot ((\mathsf{id} \times \mathsf{id}) \times (f \times g)) \\
= & \qquad \{ \ \textit{function lifting} \ \} \\
 & a_{\ulcorner f \times g \urcorner} \cdot (\mathsf{r} \times \mathsf{id})
\end{array}$$

□

LEMMA §5.37

Proof.

Associativity: We prove $\mathsf{a} : (U_p \times U_q) \times U_r \longrightarrow U_p \times (U_q \times U_r)$ is a comorphism from $((p \boxtimes q) \boxtimes r)[\mathsf{id}, \mathsf{a}]$ to $(p \boxtimes (q \boxtimes r))[\mathsf{a}, \mathsf{id}]$.

$$\begin{aligned}
& a_{(p\boxtimes(q\boxtimes r))[\mathsf{a},\mathsf{id}]} \cdot (\mathsf{a} \times \mathsf{id}) \\
= \quad & \{ \ \boxtimes \text{ and } \textit{wrapping} \text{ definitions} \ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (a_p \times (\mathsf{Bm} \cdot \delta_l \cdot (a_q \times a_r) \cdot \mathsf{m})) \cdot \mathsf{m} \cdot (\mathsf{a} \times \mathsf{a}) \\
= \quad & \{ \ \text{routine: } (\mathsf{id} \times \mathsf{m}) \cdot \mathsf{m} \cdot (\mathsf{a} \times \mathsf{a}) = \mathsf{a} \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{m} \ \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (\mathsf{id} \times \mathsf{Bm}) \cdot (\mathsf{id} \times \delta_l) \cdot (a_p \times (a_q \times a_r)) \cdot \mathsf{a} \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{m} \\
= \quad & \{ \ \mathsf{a} \text{ and } \delta_l \text{ natural (C.31)} \ \} \\
& \mathsf{Bm} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m}) \cdot \delta_l \cdot (\mathsf{id} \times \delta_l) \cdot \mathsf{a} \cdot ((a_p \times a_q) \times a_r) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{m} \\
= \quad & \{ \ \text{law (C.79)} \ \} \\
& \mathsf{Bm} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{m}) \cdot \mathsf{Ba} \cdot \delta_l \cdot (\delta_l \times \mathsf{id}) \cdot ((a_p \times a_q) \times a_r) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{m} \\
= \quad & \{ \ \text{routine: } \mathsf{m} \cdot (\mathsf{id} \times \mathsf{m}) \cdot \mathsf{a} = (\mathsf{a} \times \mathsf{a}) \cdot \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) \ \} \\
& \mathsf{B}(\mathsf{a} \times \mathsf{a}) \cdot \mathsf{Bm} \cdot \mathsf{B}(\mathsf{m} \times \mathsf{id}) \cdot \delta_l \cdot (\delta_l \times \mathsf{id}) \cdot ((a_p \times a_q) \times a_r) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{m} \\
= \quad & \{ \ \delta_l \text{ natural (C.31)} \ \} \\
& \mathsf{B}(\mathsf{a} \times \mathsf{a}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (\mathsf{Bm} \times \mathsf{id}) \cdot (\delta_l \times \mathsf{id}) \cdot ((a_p \times a_q) \times a_r) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{m} \\
= \quad & \{ \ \text{functors} \ \} \\
& \mathsf{B}(\mathsf{a} \times \mathsf{a}) \cdot \mathsf{Bm} \cdot \delta_l \cdot ((\mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m}) \times a_r) \cdot \mathsf{m} \\
= \quad & \{ \ \boxtimes \text{ and } \textit{wrapping} \text{ definitions} \ \} \\
& \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot a_{((p\boxtimes q)\boxtimes r)[\mathsf{id},\mathsf{a}]}
\end{aligned}$$

Unit: We prove $\mathsf{r} : \mathbf{1} \times U_p \longrightarrow U_p$ is comorphism from $\mathsf{idle} \boxtimes p$ to $p[\mathsf{r}, \mathsf{r}^\circ]$. As seeds are trivially preserved, this establishes equation (5.35).

$$\begin{aligned}
& a_{p[\mathsf{r},\mathsf{r}^\circ]} \cdot (\mathsf{r} \times \mathsf{id}) \\
= \quad & \{ \ \textit{wrapping} \text{ definition} \ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{r}^\circ) \cdot a_p \cdot (\mathsf{id} \times \mathsf{r}) \cdot (\mathsf{r} \times \mathsf{id}) \\
= \quad & \{ \ \text{routine: } \mathsf{r} \times \mathsf{r} = \pi_2 \cdot \mathsf{m} \ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{r}^\circ) \cdot a_p \cdot \pi_2 \cdot \mathsf{m} \\
= \quad & \{ \ \times \text{ cancellation} \ \}
\end{aligned}$$

$$\begin{array}{ll}
 & \mathsf{B}(\mathsf{id} \times \mathsf{r}^\circ) \cdot \pi_2 \cdot (\mathsf{id} \times a_p) \cdot \mathsf{m} \\
= & \qquad \{ \text{ law (C.13) } \} \\
 & \mathsf{B}(\mathsf{id} \times \mathsf{r}^\circ) \cdot \mathsf{B}\pi_2 \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{m} \\
= & \qquad \{ \text{ law (C.27) } \} \\
 & \mathsf{B}(\mathsf{id} \times \mathsf{r}^\circ) \cdot \mathsf{B}\pi_2 \cdot \delta_l \cdot (\eta \times a_p) \cdot \mathsf{m} \\
= & \qquad \{ \text{ routine: } (\mathsf{id} \times \mathsf{r}^\circ) \cdot \pi_2 = (\mathsf{r} \times \mathsf{id}) \cdot \mathsf{m} \ \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (\eta \times a_p) \cdot \mathsf{m} \\
= & \qquad \{ \ \boxtimes \text{ and } \mathsf{idle} \text{ definition } \} \\
 & \mathsf{B}(\mathsf{r} \times \mathsf{id}) \cdot a_{\mathsf{idle} \boxtimes p}
\end{array}$$

Zero: By choosing $\pi_1 : \mathbf{1} \times U_p \longrightarrow \mathbf{1}$, which clearly preserves seeds, the verification of the comorphism condition is also trivial. In fact, the task consists of proving equal two functions —$\mathsf{B}(\pi_1 \times \mathsf{id}) \cdot a_{\mathsf{nil} \boxtimes p}$ and $a_{\mathsf{nil}} \cdot (\pi_1 \times \mathsf{id})$ from $(\mathbf{1} \times U_p) \times (\emptyset \times I)$ to $\mathsf{B}((\mathbf{1} \times U_p) \times (\emptyset \times O))$ — whose domain is isomorphic to $\emptyset$ (through $\mathsf{zr} \cdot (\mathsf{id} \times \mathsf{zl}) : (\mathbf{1} \times U_p) \times (\emptyset \times I) \longrightarrow \emptyset$). Therefore, by initiality, there is only one such function, which, moreover coincides with $? \cdot \mathsf{zr} \cdot (\mathsf{id} \times \mathsf{zl})$.

Commutativity: We prove that $\mathsf{s} : U_p \times U_q \longrightarrow U_q \times U_p$ is comorphism from $p \boxtimes q$ to $q \boxtimes p[\mathsf{s}, \mathsf{s}]$. As seeds are trivially preserved, this establishes equation (5.34).

$$\begin{array}{ll}
 & a_{q \boxtimes p[\mathsf{s},\mathsf{s}]} \cdot (\mathsf{s} \times \mathsf{id}) \\
= & \qquad \{ \ \boxtimes \text{ and } \textit{wrapping} \text{ definitions } \} \\
 & \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_q \times a_p) \cdot \mathsf{m} \cdot (\mathsf{id} \times \mathsf{s}) \cdot (\mathsf{s} \times \mathsf{id}) \\
= & \qquad \{ \ \mathsf{s} \text{ natural and } \mathsf{s} \cdot \mathsf{m} = \mathsf{m} \cdot (\mathsf{s} \times \mathsf{s}) \ \} \\
 & \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_l \cdot \mathsf{s} \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \qquad \{ \ \delta_l, \delta_r \text{ interchangeable (C.76) } \} \\
 & \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \mathsf{Bs} \cdot \delta_r \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \qquad \{ \text{ routine: } \mathsf{m} \cdot \mathsf{s} = (\mathsf{s} \times \mathsf{s}) \cdot \mathsf{m} \ \} \\
 & \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{B}(\mathsf{s} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_r \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \qquad \{ \ \mathsf{B} \text{ commutative } \} \\
 & \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{B}(\mathsf{s} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \qquad \{ \ \mathsf{s} = \mathsf{s}^\circ, \boxtimes \text{ and } \textit{wrapping} \text{ definitions } \} \\
 & \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot a_{p \boxtimes q}
\end{array}$$

□

LEMMA §5.46

Proof. To complete the proof, consider the full calculation which establishes equation (5.47):

$$a_{(p\boxast p');(q\boxast q')}$$

$= \quad \{$; definition $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times a_{q\boxast q'})\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot\tau_r\cdot(a_{p\boxast p'}\times\mathsf{id})\cdot\mathsf{xr}$$

$= \quad \{$ $\boxast$ definition $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot\tau_r\cdot([\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\times\mathsf{id})\cdot((a_{p\boxplus p'}+a_{p\boxtimes p'})\times\mathsf{id})\cdot(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}$$

$= \quad \{$ routine: $(\mathsf{dr}\times\mathsf{id})\cdot\mathsf{xr}=\mathsf{dl}^{\circ}\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}$ $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot\tau_r\cdot([\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\times\mathsf{id})\cdot((a_{p\boxplus p'}+a_{p\boxtimes p'})\times\mathsf{id})\cdot\mathsf{dl}^{\circ}\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}$$

$= \quad \{$ $+$ absorption $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot\tau_r\cdot([\mathsf{B}(\mathsf{id}\times\iota_1)\cdot a_{p\boxplus p'},\mathsf{B}(\mathsf{id}\times\iota_2)\cdot a_{p\boxtimes p'}]\times\mathsf{id})\cdot\mathsf{dl}^{\circ}\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}$$

$= \quad \{$ dl° definition ($\mathsf{dl}^{\circ}=[\iota_1\times\mathsf{id},\iota_2\times\mathsf{id}]$)$\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot\tau_r\cdot([\mathsf{B}(\mathsf{id}\times\iota_1)\cdot a_{p\boxplus p'},\mathsf{B}(\mathsf{id}\times\iota_2)\cdot a_{p\boxtimes p'}]\times\mathsf{id})\cdot[\iota_1\times\mathsf{id},\iota_2\times\mathsf{id}]$$
$$\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}$$

$= \quad \{$ $+$ fusion, $+$ cancellation $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot\tau_r\cdot[(\mathsf{B}(\mathsf{id}\times\iota_1)\cdot a_{p\boxplus p'})\times\mathsf{id},(\mathsf{B}(\mathsf{id}\times\iota_2)\cdot a_{p\boxtimes p'})\times\mathsf{id}]\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}$$

$= \quad \{$ $+$ fusion $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot[\tau_r\cdot((\mathsf{B}(\mathsf{id}\times\iota_1)\cdot a_{p\boxplus p'})\times\mathsf{id}),\tau_r\cdot((\mathsf{B}(\mathsf{id}\times\iota_2)\cdot a_{p\boxtimes p'})\times\mathsf{id})]\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}$$

$= \quad \{$ τ_r natural (C.5) $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot[\mathsf{B}((\mathsf{id}\times\iota_1)\times\mathsf{id})\cdot\tau_r\cdot(a_{p\boxplus p'}\times\mathsf{id}),\mathsf{B}((\mathsf{id}\times\iota_2)\times\mathsf{id})\cdot\tau_r\cdot(a_{p\boxtimes p'}\times\mathsf{id})]$$
$$\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}$$

$= \quad \{$ $+$ absorption $\}$

$$\mu\cdot\mathsf{BBa}^{\circ}\cdot\mathsf{B}\tau_l\cdot\mathsf{B}(\mathsf{id}\times[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)])\cdot\mathsf{B}(\mathsf{id}\times(a_{q\boxplus q'}+a_{q\boxtimes q'}))\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dr})$$
$$\cdot\mathsf{Ba}\cdot\mathsf{Bxr}\cdot[\mathsf{B}((\mathsf{id}\times\iota_1)\times\mathsf{id}),\mathsf{B}((\mathsf{id}\times\iota_2)\times\mathsf{id})]\cdot(a_{p\boxplus p'}\times\mathsf{id}+\mathsf{id}\times a_{p\boxtimes p'})\cdot(\tau_r+\tau_r)$$

$$\cdot(\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ $+$ fusion, $\mathsf{a} \cdot \mathsf{xr}$ natural and $+$ absorption $\}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{q \boxplus q'} + a_{q \boxtimes q'})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr})$$
$$\cdot[\mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_1)), \mathsf{B}(\mathsf{id}(\mathsf{id} \times \iota_2))] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'})$$
$$\cdot(\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ $+$ fusion and laws (C.55), (C.56) $\}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{q \boxplus q'} + a_{q \boxtimes q'}))$$
$$\cdot[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'}) \cdot (\tau_r + \tau_r)$$
$$\cdot(\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ $+$ fusion, $+$ cancellation $\}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1 \cdot a_{q \boxplus q'}), \mathsf{B}(\mathsf{id} \times \iota_2 \cdot a_{q \boxtimes q'})]$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'}) \cdot (\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ $+$ absorption $\}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]$$
$$\cdot(\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'})) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'})$$
$$\cdot(\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ $+$ fusion, $+$ cancellation $\}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot [\mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times \iota_1)), \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times \iota_2))] \cdot (\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'}))$$
$$\cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'}) \cdot (\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ $+$ fusion $\}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot [\mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times \iota_1)), \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times \mathsf{B}(\mathsf{id} \times \iota_2))]$$
$$\cdot(\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'})) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'})$$
$$\cdot(\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ τ_l natural (C.6) and $+$ absorption $\}$

$$\mu \cdot \mathsf{BBa}^\circ \cdot [\mathsf{BB}(\mathsf{id} \times (\mathsf{id} \times \iota_1)), \mathsf{BB}(\mathsf{id} \times (\mathsf{id} \times \iota_2))] \cdot (\mathsf{B}\tau_l + \mathsf{B}\tau_l)$$
$$\cdot(\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'})) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'})$$
$$\cdot(\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$=$ $\{$ $+$ fusion $\}$

$$\mu \cdot [\mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times (\mathsf{id} \times \iota_1)), \mathsf{BBa}^\circ \cdot \mathsf{BB}(\mathsf{id} \times (\mathsf{id} \times \iota_2))] \cdot (\mathsf{B}\tau_l + \mathsf{B}\tau_l)$$
$$\cdot(\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'})) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'})$$
$$\cdot(\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr}$$

$$
\begin{array}{ll}
= & \qquad \{\ \mathsf{a}^\circ \text{ natural and } + \text{ absorption } \} \\
& \mu \cdot [\mathsf{BB}(\mathsf{id} \times \iota_1), \mathsf{BB}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{BBa}^\circ + \mathsf{BBa}^\circ) \cdot (\mathsf{B}\tau_l + \mathsf{B}\tau_l) \\
& \cdot(\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'})) \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'}) \\
& \cdot(\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \qquad \{\ + \text{ fusion, } \mu \text{ natural and } + \text{ absorption } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu + \mu) \cdot (\mathsf{B}\tau_l + \mathsf{B}\tau_l) \cdot (\mathsf{B}(\mathsf{id} \times a_{q \boxplus q'}) + \mathsf{B}(\mathsf{id} \times a_{q \boxtimes q'})) \\
& \cdot(\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (a_{p \boxplus p'} \times \mathsf{id} + \mathsf{id} \times a_{p \boxtimes p'}) \cdot (\tau_r + \tau_r) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \qquad \{\ ; \text{ definition } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{(p \boxplus p');(q \boxplus q')} + a_{(p \boxtimes p');(q \boxtimes q')}) \cdot \mathsf{dr}
\end{array}
$$

□

$\boxed{\text{LEMMA §5.47}}$

Proof. We first prove the left unit law using the direct method discussed in the main text. First note that the wiring isomorphism l_* can be defined as $\mathsf{l}_+ \cdot (\mathsf{l}_+ + \mathsf{zr})$. Unfolding the definition of $\boxast$, we arrive at

$$a_{p \boxast \mathsf{nil}} \;=\; [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{p \boxplus \mathsf{nil}} + a_{p \boxtimes \mathsf{nil}}) \cdot \mathsf{dr}$$

Next, using laws (5.20) in §26 and (5.38) in §38, $a_{p \boxplus \mathsf{nil}}$ and $a_{p \boxtimes \mathsf{nil}}$ are replaced, up to a bisimulation, by $a_{p[\mathsf{l}_+, \mathsf{l}_+{}^\circ]}$ and $a_{\mathsf{nil}[\mathsf{zr}, \mathsf{zr}^\circ]}$, respectively. Then, the resulting expression is transformed until the right hand side of the equation is reached. Thus,

$$
\begin{array}{cl}
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{p[\mathsf{l}_+, \mathsf{l}_+{}^\circ]} + a_{\mathsf{nil}[\mathsf{zr}, \mathsf{zr}^\circ]}) \cdot \mathsf{dr} \\
= & \quad \{ \ \textit{wrapping} \text{ and } \mathsf{nil} \text{ definitions} \ \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \\
 & \cdot(\mathsf{B}(\mathsf{id} \times \mathsf{l}_+{}^\circ) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+) + \mathsf{B}(\mathsf{id} \times \mathsf{zr}^\circ) \cdot \eta \cdot (\mathsf{id} \times ?) \cdot (\mathsf{id} \times \mathsf{zr})) \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ absorption} \ \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1 \cdot \mathsf{l}_+{}^\circ) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+), \mathsf{B}(\mathsf{id} \times \iota_2 \cdot \mathsf{zr}^\circ) \cdot \eta \cdot (\mathsf{id} \times ?) \cdot (\mathsf{id} \times \mathsf{zr})] \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ absorption and } + \text{ fusion} \ \} \\
 & \mathsf{B}(\mathsf{id} \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \eta \cdot (\mathsf{id} \times ?) \cdot (\mathsf{id} \times \mathsf{zr})] \cdot \mathsf{dr} \\
= & \quad \{ \ \text{corollary of initiality: } \mathsf{id} \times ? = ? \cdot \mathsf{zr} \ \} \\
 & \mathsf{B}(\mathsf{id} \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \eta \cdot ? \cdot \mathsf{zr} \cdot (\mathsf{id} \times \mathsf{zr})] \cdot \mathsf{dr} \\
= & \quad \{ \ \text{initiality: } f \cdot ? = ? \ \} \\
 & \mathsf{B}(\mathsf{id} \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+), \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot ? \cdot \mathsf{zr} \cdot (\mathsf{id} \times \mathsf{zr})] \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ fusion} \ \} \\
 & \mathsf{B}(\mathsf{id} \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot [\mathsf{id} \times \mathsf{l}_+, ? \cdot \mathsf{zr} \cdot (\mathsf{id} \times \mathsf{zr})] \cdot \mathsf{dr} \\
= & \quad \{ \ \mathsf{l}_+{}^\circ = \iota_1 \text{ and } \mathsf{id} \times ? = ? \cdot \mathsf{zr} \ \} \\
 & \mathsf{B}(\mathsf{id} \times ((\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{l}_+{}^\circ)) \cdot a_p \cdot [\mathsf{id} \times \mathsf{l}_+, \mathsf{id} \times ? \cdot \mathsf{zr}] \cdot \mathsf{dr} \\
= & \quad \{ \ \text{law (C.49)} \ \} \\
 & \mathsf{B}(\mathsf{id} \times ((\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{l}_+{}^\circ)) \cdot a_p \cdot (\mathsf{id} \times [\mathsf{l}_+, ? \cdot \mathsf{zr}]) \\
= & \quad \{ \ \text{routine: } [\mathsf{l}_+, ? \cdot \mathsf{zr}] = \mathsf{l}_+ \cdot (\mathsf{l}_+ + \mathsf{zr}) \ \} \\
 & \mathsf{B}(\mathsf{id} \times ((\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{l}_+{}^\circ)) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+ \cdot (\mathsf{l}_+ + \mathsf{zr})) \\
= & \quad \{ \ \textit{wrapping} \text{ definition} \ \} \\
 & a_{p[\mathsf{l}_*, \mathsf{l}_*{}^\circ]}
\end{array}
$$

A proof 'from first principles' proceeds by the identification of a Cp morphism from $a_{p \boxast \mathsf{nil}}$ to $a_{p[\mathsf{l}_*, \mathsf{l}_*{}^\circ]}$. Clearly $\mathsf{l} : U_p \times \mathbf{1} \longrightarrow U_p$ would do the job. We have to check it preserves seeds, which is trivial, and satisfies the comorphism condition expressed by the commutativity of the following diagram:

$$\begin{tikzcd}
(U_p \times \mathbf{1}) \times ((I + \emptyset) + I \times \emptyset) \arrow[r, "\mathsf{l} \times \mathsf{id}"] \arrow[d, "a_{p \boxast \mathsf{nil}}"'] & U_p \times ((I + \emptyset) + I \times \emptyset) \arrow[d, "a_{p[\mathsf{l}_*, \mathsf{l}_*{}^\circ]}"] \\
\mathsf{B}((U_p \times \mathbf{1}) \times ((O + \emptyset) + O \times \emptyset)) \arrow[r, "\mathsf{B}(\mathsf{l} \times \mathsf{id})"] & \mathsf{B}\ (U_p \times ((O + \emptyset) + O \times \emptyset))
\end{tikzcd}$$

The verification of this diagram follows the same lines of the direct proof above, but for the explicit presence of comorphisms which witness the ⊞ and ⊠ laws re-used. Notice that, in this case, they are also l for both tensors: all the reader has to do is to check the beginning of the proofs of laws (5.20) in §26 and (5.38) in §38, respectively. We 'repeat' the proof to check the details:

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot a_{p \boxast \mathsf{nil}}$$

= { $\boxast$ definition }

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{p \boxplus \mathsf{nil}} + a_{p \boxtimes \mathsf{nil}}) \cdot \mathsf{dr}$$

= { laws (5.20) in §26 and (5.38) in §38, both witnessed by l }

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]$$
$$\cdot(\mathsf{B}(\mathsf{l}^\circ \times \mathsf{id}) \cdot a_{p[\mathsf{l}_+, \mathsf{l}_+{}^\circ]} \cdot (\mathsf{l} \times \mathsf{id}) + \mathsf{B}(\mathsf{l}^\circ \times \mathsf{id}) \cdot a_{\mathsf{nil}[\mathsf{zr}, \mathsf{zr}^\circ]} \cdot (\mathsf{l} \times \mathsf{id})) \cdot \mathsf{dr}$$

= { *wrapping* and nil definitions }

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]$$
$$\cdot(\mathsf{B}(\mathsf{l}^\circ \times \mathsf{l}_+{}^\circ) \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+) + \mathsf{B}(\mathsf{l}^\circ \times \mathsf{zr}^\circ) \cdot \eta \cdot (\mathsf{id} \times ?) \cdot (\mathsf{l} \times \mathsf{zr})) \cdot \mathsf{dr}$$

= { $+$ absorption }

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{l}^\circ \times \iota_1 \cdot \mathsf{l}_+{}^\circ) \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+), \mathsf{B}(\mathsf{l}^\circ \times \iota_2 \cdot \mathsf{zr}^\circ) \cdot \eta \cdot (\mathsf{id} \times ?) \cdot (\mathsf{l} \times \mathsf{zr})] \cdot \mathsf{dr}$$

= { $+$ absorption and $+$ fusion }

$$\mathsf{B}(\mathsf{l} \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{l}^\circ \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ)$$
$$\cdot[\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \eta \cdot (\mathsf{id} \times ?) \cdot (\mathsf{l} \times \mathsf{zr})] \cdot \mathsf{dr}$$

= { l isomorphism and, as a corollary of initiality, $\mathsf{id} \times ? = ? \cdot \mathsf{zr}$ }

$$\mathsf{B}(\mathsf{id} \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \eta \cdot ? \cdot \mathsf{zr} \cdot (\mathsf{l} \times \mathsf{zr})] \cdot \mathsf{dr}$$

= { initiality: $f \cdot ? = ?$ }

$$\mathsf{B}(\mathsf{id} \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot (\mathsf{l} \times \mathsf{l}_+), \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot ? \cdot \mathsf{zr} \cdot (\mathsf{l} \times \mathsf{zr})] \cdot \mathsf{dr}$$

= { $+$ fusion }

$$\mathsf{B}(\mathsf{id} \times (\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_p \cdot [\mathsf{l} \times \mathsf{l}_+, ? \cdot \mathsf{zr} \cdot (\mathsf{l} \times \mathsf{zr})] \cdot \mathsf{dr}$$

$$= \quad \{ \ \mathsf{l}_+{}^\circ = \iota_1 \text{ and } \mathsf{id} \times ? = ? \cdot \mathsf{zr} \ \}$$

$$\mathsf{B}(\mathsf{id} \times ((\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{l}_+{}^\circ)) \cdot a_p \cdot [\mathsf{l} \times \mathsf{l}_+, \mathsf{l} \times ? \cdot \mathsf{zr}] \cdot \mathsf{dr}$$

$$= \quad \{ \ \text{law (C.49)} \ \}$$

$$\mathsf{B}(\mathsf{id} \times ((\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{l}_+{}^\circ)) \cdot a_p \cdot (\mathsf{l} \times [\mathsf{l}_+, ? \cdot \mathsf{zr}])$$

$$= \quad \{ \ \text{routine: } [\mathsf{l}_+, ? \cdot \mathsf{zr}] = \mathsf{l}_+ \cdot (\mathsf{l}_+ + \mathsf{zr}) \ \}$$

$$\mathsf{B}(\mathsf{id} \times ((\mathsf{l}_+{}^\circ + \mathsf{zr}^\circ) \cdot \mathsf{l}_+{}^\circ)) \cdot a_p \cdot (\mathsf{id} \times \mathsf{l}_+ \cdot (\mathsf{l}_+ + \mathsf{zr})) \cdot (\mathsf{l} \times \mathsf{id})$$

$$= \quad \{ \ \textit{wrapping} \text{ definition} \ \}$$

$$a_{p[\mathsf{r}_*, \mathsf{r}_*{}^\circ]} \cdot (\mathsf{l} \times \mathsf{id})$$

As remarked earlier on, proofs of the remaining laws follow a similar pattern: the ‘hard’ work has already been done when dealing with $\boxplus$ and $\boxtimes$. Therefore, we verify here, again using the ‘direct’ procedure, the commutativity law (5.50) which requires B to be a commutative monad. First note the wiring isomorphism is

$$\mathsf{s}_* \ = \ \mathsf{s}_+ + \mathsf{s}$$

Then, by definition of $\boxast$,

$$a_{(q \boxast p)[\mathsf{s}_*, \mathsf{s}_*]} \ = \ ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)](a_{(q \boxplus p)} + a_{(q \boxtimes p)}) \cdot \mathsf{dr})[\mathsf{s}_*, \mathsf{s}_*]$$

Now, using laws (5.21) in §26 and (5.34) in §38, $a_{\,q \boxplus p}$ and $a_{q \boxtimes p}$ are replaced, up to bisimilarity, by $a_{(p \boxplus q)[\mathsf{s}_+, \mathsf{s}_+]}$ and $a_{(p \boxtimes q)[\mathsf{s}, \mathsf{s}]}$, respectively. Then, the resulting expression is transformed until the right hand side of the equation is reached. Thus,

$$([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)](a_{(p \boxplus q)[\mathsf{s}_+, \mathsf{s}_+]} + a_{(p \boxtimes q)[\mathsf{s}, \mathsf{s}]}) \cdot \mathsf{dr})[\mathsf{s}_*, \mathsf{s}_*]$$

$$= \quad \{ \ \textit{wrapping} \text{ definition and functors} \ \}$$

$$\begin{array}{l} \mathsf{B}(\mathsf{id} \times (\mathsf{s}_+ + \mathsf{s})) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \\ \cdot (\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_{(p \boxplus q)} \cdot (\mathsf{id} \times \mathsf{s}_+) + \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot a_{(p \boxtimes q)} \cdot (\mathsf{id} \times \mathsf{s})) \cdot \mathsf{dr} \cdot (\mathsf{id} \times (\mathsf{s}_+ + \mathsf{s})) \end{array}$$

$$= \quad \{ \ \mathsf{dr} \text{ natural} \ \}$$

$$\begin{array}{l} \mathsf{B}(\mathsf{id} \times (\mathsf{s}_+ + \mathsf{s})) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \\ \cdot (\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_{(p \boxplus q)} \cdot (\mathsf{id} \times \mathsf{s}_+) + \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot a_{(p \boxtimes q)} \cdot (\mathsf{id} \times \mathsf{s})) \cdot (\mathsf{id} \times \mathsf{s}_+ + \mathsf{id} \times \mathsf{s}) \\ \cdot \mathsf{dr} \end{array}$$

$$= \quad \{ \ \mathsf{s}_+, \mathsf{s} \text{ isomorphisms} \ \}$$

$$\begin{array}{l} \mathsf{B}(\mathsf{id} \times (\mathsf{s}_+ + \mathsf{s})) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_{(p \boxplus q)} + \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot a_{(p \boxtimes q)}) \\ \cdot \mathsf{dr} \end{array}$$

$$= \quad \{ \ + \text{ fusion and } + \text{ absorption} \ \}$$

$$[\mathsf{B}(\mathsf{id} \times \iota_1 \cdot \mathsf{s}_+), \mathsf{B}(\mathsf{id} \times \iota_2 \cdot \mathsf{s})] \cdot (\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_{(p \boxplus q)} + \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot a_{(p \boxtimes q)}) \cdot \mathsf{dr}$$

$=$ { $+$ absorption }

$$[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]$$
$$\cdot(\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_{(p \boxplus q)} + \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot a_{(p \boxtimes q)}) \cdot \mathsf{dr}$$

$=$ { s_+, s isomorphisms }

$$[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{(p \boxplus q)} + a_{(p \boxtimes q)}) \cdot \mathsf{dr}$$

$=$ { $\boxast$ definition }

$$a_{(p \boxast q)}$$

□

$\boxed{\text{LEMMA §5.52}}$

Proof. Let $h : p \longrightarrow q$ be an arrow in $\mathsf{Cp}(K, K)$. Then,

$$\begin{aligned}
& a_{q\Lsh} \cdot (h \times \mathsf{id}) \\
= \quad & \quad \{\ q \Lsh \text{ definition } \} \\
& \mu \cdot \mathsf{B}a_q \cdot a_q \cdot (h \times \mathsf{id}) \\
= \quad & \quad \{\ \text{assumption: } h : p \longrightarrow q\ \} \\
& \mu \cdot \mathsf{B}a_q \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_p \\
= \quad & \quad \{\ \text{assumption: } h : p \longrightarrow q\ \} \\
& \mu \cdot \mathsf{B}(\mathsf{B}(h \times \mathsf{id}) \cdot a_p) \cdot a_p \\
= \quad & \quad \{\ \mu \text{ natural (C.16) } \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot \mu \cdot \mathsf{B}a_p \cdot a_p \\
= \quad & \quad \{\ p \Lsh \text{ definition } \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot a_{p\Lsh}
\end{aligned}$$

Similarly, for an arrow $h : p \longrightarrow q$ in $\mathsf{Cp}(I + K, O + K)$,

$$\begin{aligned}
& a_{q\Lsh_Z} \cdot (h \times \mathsf{id}) \\
= \quad & \quad \{\ q \Lsh_Z \text{ definition } \} \\
& \mu \cdot \mathsf{B}\triangledown \cdot \mathsf{B}(\eta + a_q) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot a_q \cdot (h \times \mathsf{id}) \\
= \quad & \quad \{\ \text{assumption: } h : p \longrightarrow q\ \} \\
& \mu \cdot \mathsf{B}\triangledown \cdot \mathsf{B}(\eta + a_q) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_p \\
= \quad & \quad \{\ \mathsf{dr} \text{ natural } \} \\
& \mu \cdot \mathsf{B}\triangledown \cdot \mathsf{B}(\eta + a_q) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{B}(h \times \mathsf{id} + h \times \mathsf{id}) \cdot \mathsf{Bdr} \cdot a_p \\
= \quad & \quad \{\ \text{functors } \} \\
& \mu \cdot \mathsf{B}\triangledown \cdot \mathsf{B}(\eta \cdot (h \times \mathsf{id}) + a_q \cdot (h \times \mathsf{id})) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot a_p \\
= \quad & \quad \{\ \text{assumption: } h : p \longrightarrow q \text{ and } \eta \text{ natural (C.17) } \} \\
& \mu \cdot \mathsf{B}\triangledown \cdot \mathsf{B}(\mathsf{B}(h \times \mathsf{id}) \cdot \eta + \mathsf{B}(h \times \mathsf{id}) \cdot a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot a_p \\
= \quad & \quad \{\ \triangledown \text{ natural } \} \\
& \mu \cdot \mathsf{BB}(h \times \mathsf{id}) \cdot \mathsf{B}\triangledown \cdot \mathsf{B}(\eta + a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot a_p \\
= \quad & \quad \{\ \mu \text{ natural (C.16) } \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot \mu \cdot \mathsf{B}\triangledown \cdot \mathsf{B}(\eta + a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot a_p
\end{aligned}$$

$=$ $\{\ p\Lsh_Z$ definition $\}$

$$\mathsf{B}(h \times \mathsf{id}) \cdot a_{p\Lsh_Z}$$

□

$\boxed{\text{LEMMA §5.53}}$

Proof. Both equations (5.55) and (5.56) are strict equalities. In fact,

$$\begin{array}{cl}
 & a_{\ulcorner f \urcorner \Lsh} \\
= & \quad \{ \text{ feedback and \textit{function lifting} definitions } \} \\
 & \mu \cdot \mathsf{B}(\eta \cdot (\mathsf{id} \times f)) \cdot \eta \cdot (\mathsf{id} \times f) \\
= & \quad \{ \text{ monad axiom (C.14) } \} \\
 & \mathsf{B}(\mathsf{id} \times f) \cdot \eta \cdot (\mathsf{id} \times f) \\
= & \quad \{ \ \eta \text{ natural (C.17) } \} \\
 & \eta \cdot (\mathsf{id} \times (f \cdot f)) \\
= & \quad \{ \text{ \textit{function lifting} definition } \} \\
 & a_{\ulcorner f \cdot f \urcorner}
\end{array}$$

And, similarly,

$$\begin{array}{cl}
 & a_{\ulcorner f \urcorner \Lsh_Z} \\
= & \quad \{ \text{ \textit{partial feedback} and \textit{function lifting} definitions } \} \\
 & \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + \eta) \cdot \mathsf{B}(\mathsf{id} + \mathsf{id} \times f) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot \eta \cdot (\mathsf{id} \times f) \\
= & \quad \{ \ \nabla \text{ and } \mathsf{dr} \text{ natural } \} \\
 & \mu \cdot \mathsf{B}\eta \cdot \mathsf{B}\nabla \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{id} + f)) \cdot \mathsf{B}(\mathsf{id} \times (\iota_1 + \iota_2)) \cdot \eta \cdot (\mathsf{id} \times f) \\
= & \quad \{ \text{ monad axiom (C.14) } \} \\
 & \mathsf{B}\nabla \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{id} + f)) \cdot \mathsf{B}(\mathsf{id} \times (\iota_1 + \iota_2)) \cdot \eta \cdot (\mathsf{id} \times f) \\
= & \quad \{ \text{ routine: } \nabla \cdot \mathsf{dr} = \mathsf{id} \times \nabla \ \} \\
 & \mathsf{B}(\mathsf{id} \times \nabla) \cdot \mathsf{B}(\mathsf{id} \times (\mathsf{id} + f)) \cdot \mathsf{B}(\mathsf{id} \times (\iota_1 + \iota_2)) \cdot \eta \cdot (\mathsf{id} \times f) \\
= & \quad \{ \ \eta \text{ natural (C.17) } \} \\
 & \eta \cdot (\mathsf{id} \times \nabla) \cdot (\mathsf{id} \times (\mathsf{id} + f)) \cdot (\mathsf{id} \times (\iota_1 + \iota_2)) \cdot (\mathsf{id} \times f) \\
= & \quad \{ \text{ functors and } \nabla \text{ definition } \} \\
 & \eta \cdot (\mathsf{id} \times [\mathsf{id}, \mathsf{id}] \cdot (\iota_1 + f \cdot \iota_2) \cdot f) \\
= & \quad \{ \ + \text{ absorption } \} \\
 & \eta \cdot (\mathsf{id} \times [\iota_1, f \cdot \iota_2] \cdot f) \\
= & \quad \{ \text{ \textit{function lifting} } \} \\
 & a_{\ulcorner [\iota_1, f \cdot \iota_2] \cdot f \urcorner}
\end{array}$$

□

LEMMA §5.54

Proof. Both equations are strict equalities. To establish (5.57) consider

$$a_{p[\mathsf{r}_+,\mathsf{r}_+{}^\circ]\Lsh_Z[\mathsf{r}_+{}^\circ,\mathsf{r}_+]}$$

$=$ { *wrapping* definition }

$$\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot a_{p[\mathsf{r}_+,\mathsf{r}_+{}^\circ]\Lsh_Z}\cdot(\mathsf{id}\times\mathsf{r}_+{}^\circ)$$

$=$ { *partial feedback* definition }

$$\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot\mu\cdot\mathsf{B}\nabla\cdot\mathsf{B}(\eta+a_{p[\mathsf{r}_+,\mathsf{r}_+{}^\circ]})\cdot\mathsf{B}(\mathsf{id}\times\iota_1+\mathsf{id}\times\iota_2)\cdot\mathsf{Bdr}\cdot a_{p[\mathsf{r}_+,\mathsf{r}_+{}^\circ]}$$
$$\cdot(\mathsf{id}\times\mathsf{r}_+{}^\circ)$$

$=$ { *wrapping* definition }

$$\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot\mu\cdot\mathsf{B}\nabla\cdot\mathsf{B}(\eta+\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p\cdot(\mathsf{id}\times\mathsf{r}_+))\cdot\mathsf{B}(\mathsf{id}\times\iota_1+\mathsf{id}\times\iota_2)\cdot\mathsf{Bdr}$$
$$\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p\cdot(id\times\mathsf{r}_+)\cdot(\mathsf{id}\times\mathsf{r}_+{}^\circ)$$

$=$ { r_+ isomorphism and $\mathsf{r}_+{}^\circ=\iota_2 : X\longrightarrow\emptyset+X$ }

$$\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot\mu\cdot\mathsf{B}\nabla\cdot\mathsf{B}(\eta+\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p)\cdot\mathsf{B}(\mathsf{id}\times\iota_1+\mathsf{id})\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$=$ { functors }

$$\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot\mu\cdot\mathsf{B}\nabla\cdot\mathsf{B}(\eta+\mathsf{id})\cdot\mathsf{B}(\mathsf{id}+\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ))\cdot\mathsf{B}(\mathsf{id}+a_p)\cdot\mathsf{B}(\mathsf{id}\times\iota_1+\mathsf{id})$$
$$\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$=$ { initiality: $(?\cdot\mathsf{zr})=\eta\cdot(\mathsf{id}\times\iota_1) : U_p\times\emptyset\longrightarrow\mathsf{B}(U_p\times(\emptyset+Z))$ }

$$\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot\mu\cdot\mathsf{B}\nabla\cdot\mathsf{B}(?+\mathsf{id})\cdot\mathsf{B}(\mathsf{zr}+\mathsf{id})\cdot\mathsf{B}(\mathsf{id}+\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ))\cdot\mathsf{B}(\mathsf{id}+a_p)\cdot\mathsf{Bdr}$$
$$\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$=$ { ∇ definition and $+$ absorption }

$$\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot\mu\cdot\mathsf{B}[?,\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)]\cdot\mathsf{B}(\mathsf{zr}+\mathsf{id})\cdot\mathsf{B}(\mathsf{id}+a_p)\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$=$ { μ natural (C.16) }

$$\mu\cdot\mathsf{BB}(\mathsf{id}\times\mathsf{r}_+)\cdot\mathsf{B}[?,\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)]\cdot\mathsf{B}(\mathsf{zr}+\mathsf{id})\cdot\mathsf{B}(\mathsf{id}+a_p)\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$=$ { $+$ fusion }

$$\mu\cdot\mathsf{B}[\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot ?,\mathsf{B}(\mathsf{id}\times\mathsf{r}_+)\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)]\cdot\mathsf{B}(\mathsf{zr}+\mathsf{id})\cdot\mathsf{B}(\mathsf{id}+a_p)\cdot\mathsf{Bdr}$$
$$\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$=$ { r_+ isomorphism and initiality }

$$\mu\cdot\mathsf{B}[?,\mathsf{id}]\cdot\mathsf{B}(\mathsf{zr}+\mathsf{id})\cdot\mathsf{B}(\mathsf{id}+a_p)\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$=$ { $\mathsf{r}_+=[?_X,\mathsf{id}_X] : \emptyset+X\longrightarrow X$ }

$$\mu\cdot\mathsf{Br}_+\cdot\mathsf{B}(\mathsf{zr}+\mathsf{id})\cdot\mathsf{B}(\mathsf{id}+a_p)\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\mathsf{r}_+{}^\circ)\cdot a_p$$

$$= \qquad \{ \ \mathsf{r}_+ \text{ natural} \ \}$$

$$\mu \cdot \mathsf{B}a_p \cdot \mathsf{Br}_+ \cdot \mathsf{B}(\mathsf{zr} + \mathsf{id}) \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{r}_+{}^\circ) \cdot a_p$$

$$= \qquad \{ \ \text{routine: } \mathsf{r}_+{}^\circ = (\mathsf{zr} + \mathsf{id}) \cdot \mathsf{dr} \cdot (\mathsf{id} \times \mathsf{r}_+{}^\circ \ \}$$

$$\mu \cdot \mathsf{B}a_p \cdot \mathsf{Br}_+ \cdot \mathsf{Br}_+{}^\circ \cdot a_p$$

$$= \qquad \{ \ \mathsf{r}_+ \text{ isomorphism} \ \}$$

$$\mu \cdot \mathsf{B}a_p \cdot a_p$$

$$= \qquad \{ \ p \Lsh \text{ definition} \ \}$$

$$a_{p\Lsh}$$

Law (5.58) is proved as follows,

$$a_{q \boxplus p \Lsh}$$

$$= \qquad \{ \ \boxplus \text{ and feedback definitions} \ \}$$

$$[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (\mathsf{id} + \mathsf{id} \times \mu) \cdot (\mathsf{id} + (\mathsf{id} \times \mathsf{B}a_p))$$
$$\cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$$= \qquad \{ \ \text{functors and } \mu \text{ strong (C.21)} \ \}$$

$$[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} \cdot \tau_r + \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_l \cdot (\mathsf{id} \times \mathsf{B}a_p))$$
$$\cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$$= \qquad \{ \ \mu \text{ and } \tau_l \text{ natural (C.16) and (C.6)} \ \}$$

$$[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} \cdot \tau_r + \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \tau_l)$$
$$\cdot a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$$= \qquad \{ \ \text{functors and } \mathsf{a} \text{ isomorphism} \ \}$$

$$[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba}) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ)$$
$$\cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$$= \qquad \{ \ \star \ \}$$

$$\mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\mathsf{id} + \mathsf{B}(\mathsf{id} \times \iota_2)) \cdot \mathsf{B}(\mathsf{id} + \mathsf{Ba}^\circ) \cdot \mathsf{B}(\mathsf{id} + \tau_l) \cdot \mathsf{B}(\mathsf{id} + \mathsf{id} \times a_p) \cdot \mathsf{B}(\mathsf{id} + \mathsf{a})$$
$$\cdot \mathsf{B}(\eta + \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id}) \cdot \mathsf{Bdr} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l)$$
$$\cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$$= \qquad \{ \ \text{functors} \ \}$$

$$\mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\mathsf{id} + (\mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a})) \cdot \mathsf{B}(\eta + \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id}) \cdot \mathsf{Bdr}$$
$$\cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}$$

$$= \qquad \{ \ + \text{ cancellation} \ \}$$

$$\mu \cdot \mathsf{B}\nabla$$

$$
\begin{array}{ll}
& \cdot \mathsf{B}(\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_q \times \mathsf{id}) \cdot \mathsf{xr}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a}] \cdot \iota_2) \\
& \cdot \mathsf{B}(\eta + \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id}) \cdot \mathsf{Bdr} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \\
& \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ + \text{absorption and law (C.56)} \ \} \\
& \mu \cdot \mathsf{B}\nabla \\
& \cdot \mathsf{B}(\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} \cdot \tau_r \cdot (a_q \times \mathsf{id}) \cdot \mathsf{xr} + \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a})) \\
& \cdot \mathsf{B}(\mathsf{id} + \mathsf{dr} \cdot (\mathsf{id} \times \iota_2)) \cdot \mathsf{B}(\eta + \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id}) \cdot \mathsf{Bdr} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \\
& \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ \text{functors} \ \} \\
& \mu \cdot \mathsf{B}\nabla \ \cdot \\
& \mathsf{B}(\mathsf{id} + [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}) \\
& \cdot \mathsf{B}(\eta + \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \\
& \cdot (\tau_r + \tau_l) \cdot (a_q \times \mathsf{id} + \mathsf{id} \times a_p) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ \boxplus \ \text{definition} \ \} \\
& \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + a_{q \boxplus p}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot a_{q \boxplus p} \\
= & \qquad \{ \ \textit{partial feedback} \ \text{definition} \ \} \\
& a_{q \boxplus p \Lsh z}
\end{array}
$$

The step marked with a $\star$ above is justified by the following calculation:

$$
\begin{array}{ll}
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{id} + \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba}) \\
= & \qquad \{ \ \text{monad unit (C.14)} \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mu + \mu) \cdot (\mathsf{B}\eta + \mathsf{id}) \cdot (\mathsf{id} + \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba}) \\
= & \qquad \{ \ + \text{absorption} \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mu, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mu] \cdot (\mathsf{B}\eta + \mathsf{id}) \cdot (\mathsf{id} + \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba}) \\
= & \qquad \{ \ \mu \ \text{natural (C.16)} \ \} \\
& [\mu \cdot \mathsf{BB}(\mathsf{id} \times \iota_1), \mu \cdot \mathsf{BB}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}\eta + \mathsf{id}) \cdot (\mathsf{id} + \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba}) \\
= & \qquad \{ \ + \text{fusion} \ \} \\
& \mu \cdot [\mathsf{BB}(\mathsf{id} \times \iota_1), \mathsf{BB}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}\eta + \mathsf{id}) \cdot (\mathsf{id} + \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba}) \\
= & \qquad \{ \ + \text{absorption} \ \} \\
& \mu \cdot [\mathsf{BB}(\mathsf{id} \times \iota_1) \cdot \mathsf{B}\eta, \mathsf{BB}(\mathsf{id} \times \iota_2) \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba}] \\
= & \qquad \{ \ \text{functors and } \eta \text{ natural (C.17)} \} \\
& \mu \cdot [\mathsf{B}(\eta \cdot (\mathsf{id} \times \iota_1)), \mathsf{B}(\mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a})]
\end{array}
$$

$$
\begin{array}{ll}
= & \quad \{ \ + \text{cancellation} \ \} \\
 & \mu \cdot [\mathsf{B}[\eta \cdot (\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a}] \cdot \mathsf{B}\iota_1, \mathsf{B}[\cdots, \cdots] \cdot \mathsf{B}\iota_1] \\
= & \quad \{ \ + \text{fusion} \ \} \\
 & \mu \cdot \mathsf{B}[\eta \cdot (\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a}] \cdot [\mathsf{B}\iota_1, \mathsf{B}\iota_2] \\
= & \quad \{ \ \text{laws (C.55) and (C.56)} \ \} \\
 & \mu \cdot \mathsf{B}[\eta \cdot (\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a}] \\
 & \cdot [\mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \iota_2)] \\
= & \quad \{ \ \nabla \text{ definition and } + \text{ absorption, } + \text{ fusion} \ \} \\
 & \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\mathsf{id} + \mathsf{B}(\mathsf{id} \times \iota_2)) \cdot \mathsf{B}(\mathsf{id} + \mathsf{Ba}^\circ) \cdot \mathsf{B}(\mathsf{id} + \tau_l) \cdot \mathsf{B}(\mathsf{id} + \mathsf{id} \times a_p) \cdot \mathsf{B}(\mathsf{id} + \mathsf{a}) \\
 & \cdot \mathsf{B}(\eta + \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id}) \cdot \mathsf{Bdr} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]
\end{array}
$$

□

LEMMA §5.57

Proof. Equality (5.61) is established as follows:

$$
\begin{array}{ll}
 & a_{r[i,i^\circ]\Lsh} \\
= & \qquad \{ \ \textit{feedback} \text{ and } \textit{wrapping} \text{ definitions} \ \} \\
 & \mu \cdot \mathsf{BB}(\mathsf{id} \times i^\circ) \cdot \mathsf{B}a_p \cdot \mathsf{B}(\mathsf{id} \times i) \cdot \mathsf{B}(\mathsf{id} \times i^\circ) \cdot a_p \cdot (\mathsf{id} \times i) \\
= & \qquad \{ \ i \text{ isomorphism} \ \} \\
 & \mu \cdot \mathsf{BB}(\mathsf{id} \times i^\circ) \cdot \mathsf{B}a_p \cdot a_p \cdot (\mathsf{id} \times i) \\
= & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
 & \mathsf{B}(\mathsf{id} \times i^\circ) \cdot \mu \cdot \mathsf{B}a_p \cdot a_p \cdot (\mathsf{id} \times i) \\
= & \qquad \{ \ \textit{feedback} \text{ and } \textit{wrapping} \text{ definitions} \ \} \\
 & a_{r\Lsh[i,i^\circ]}
\end{array}
$$

Next consider equality (5.62):

$$
\begin{array}{ll}
 & a_{(p \boxplus q)\Lsh} \\
= & \qquad \{ \ \textit{feedback} \text{ and } \boxplus \text{ definitions} \ \} \\
 & \mu \cdot \mathsf{B}[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot \mathsf{B}(\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot \mathsf{B}(\tau_r + \tau_l) \cdot \mathsf{B}(a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
 & \cdot \mathsf{B}(\mathsf{xr} + \mathsf{a}) \cdot \mathsf{Bdr} \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
 & \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ + \text{ absorption}, + \text{ fusion} \ \} \\
 & \mu \cdot \mathsf{B}[\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_q) \cdot \mathsf{a}] \\
 & \cdot [\mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
 & \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ \text{laws (C.55) and (C.56)} \ \} \\
 & \mu \cdot \mathsf{B}[\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \tau_l \cdot (\mathsf{id} \times a_q) \cdot \mathsf{a}] \\
 & \cdot [\mathsf{B}\iota_1, \mathsf{B}\iota_2] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ + \text{ fusion}, + \text{ cancellation} \ \} \\
 & \mu \cdot [\mathsf{BB}(\mathsf{id} \times \iota_1) \cdot \mathsf{BBxr} \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(a_p \times \mathsf{id}) \cdot \mathsf{Bxr}, \mathsf{BB}(\mathsf{id} \times \iota_2) \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{Ba}] \\
 & \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ + \text{ fusion and } \mu \text{ natural (C.16)} \ \} \\
 & [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{Bxr} \cdot \mu \cdot \mathsf{B}\tau_r \cdot \mathsf{B}(a_p \times \mathsf{id}) \cdot \mathsf{Bxr}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{Ba}^\circ \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{Ba}] \\
 & \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \qquad \{ \ + \text{ absorption} \ \}
\end{array}
$$

$$
\begin{array}{cl}
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\mu + \mu) \cdot (\mathsf{B}\tau_r + \mathsf{B}\tau_l) \\
& \cdot(\mathsf{B}(a_p \times \mathsf{id}) + \mathsf{B}(\mathsf{id} \times a_q)) \cdot (\mathsf{Bxr} + \mathsf{Ba}) \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \\
& \cdot(\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \mathsf{xr} = \mathsf{xr}^\circ \text{ and } \mathsf{a} \text{ isomorphism } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\mu + \mu) \cdot (\mathsf{B}\tau_r + \mathsf{B}\tau_l) \\
& \cdot(\mathsf{B}(a_p \times \mathsf{id}) + \mathsf{B}(\mathsf{id} \times a_q)) \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \tau_r, \tau_l \text{ natural (C.5) and (C.6) } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\mu + \mu) \cdot (\mathsf{B}\tau_r + \mathsf{B}\tau_l) \\
& \cdot(\tau_r + \tau_l) \cdot (\mathsf{B}a_p \times \mathsf{id} + \mathsf{id} \times \mathsf{B}a_q) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \mu \text{ strong (C.19) and (C.21) } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{Bxr} + \mathsf{Ba}^\circ) \cdot (\tau_r + \tau_l) \cdot (\mu \times \mathsf{id} + \mathsf{id} \times \mu) \\
& \cdot(\mathsf{B}a_p \times \mathsf{id} + \mathsf{id} \times \mathsf{B}a_q) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \\
= & \quad \{ \ \textit{feedback} \text{ and } \boxplus \text{ definitions } \} \\
& a_{p\Lsh \boxplus q\Lsh}
\end{array}
$$

The somewhat simpler reasoning below establishes equality (5.63). The proof of (5.64) is a combination of the proof arguments for the two previous cases, as usual.

$$
\begin{array}{cl}
& a_{(p \boxtimes q)\Lsh} \\
= & \quad \{ \ \textit{feedback} \text{ and } \boxtimes \text{ definitions } \} \\
& \mu \cdot \mathsf{BBm} \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(a_p \times a_q) \cdot \mathsf{Bm} \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \quad \{ \ \mathsf{m} = \mathsf{m}^\circ \ \} \\
& \mu \cdot \mathsf{BBm} \cdot \mathsf{B}\delta_l \cdot \mathsf{B}(a_p \times a_q) \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \quad \{ \ \delta_l \text{ natural (C.31) } \} \\
& \mu \cdot \mathsf{BBm} \cdot \mathsf{B}\delta_l \cdot \delta_l \cdot (\mathsf{B}a_p \times \mathsf{B}a_q) \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \quad \{ \ \mu \text{ natural (C.16) } \} \\
& \mathsf{Bm} \cdot \mu \cdot \mathsf{B}\delta_l \cdot \delta_l \cdot (\mathsf{B}a_p \times \mathsf{B}a_q) \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \quad \{ \ \text{law (C.75), which requires } \mathsf{B} \text{ to be commutative } \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (\mu \times \mu) \cdot (\mathsf{B}a_p \times \mathsf{B}a_q) \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= & \quad \{ \ \textit{feedback} \text{ and } \boxtimes \text{ definitions } \} \\
& a_{p\Lsh \boxtimes q\Lsh}
\end{array}
$$

□

LEMMA §5.58

Proof. The lemma relates partial feedback with wrapping and ⊞. The proof arguments are similar to the ones followed in the corresponding proofs for the total feedback. This is illustrated below in the detailed proof of (5.65).

$$a_{p\Lsh_K[f+i,g+i^\circ]}$$

$$= \qquad \{ \textit{ feedback} \text{ and } \textit{wrapping} \text{ definitions } \}$$

$$\mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot a_p \cdot (\mathsf{id} \times (f + i))$$

$$= \qquad \{ \ \mu \text{ and } \nabla \text{ natural } \}$$

$$\mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\mathsf{B}(\mathsf{id} \times (g + i^\circ)) + \mathsf{B}(\mathsf{id} \times (g + i^\circ))) \cdot \mathsf{B}(\eta + a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2)$$
$$\cdot \mathsf{Bdr} \cdot a_p \cdot (\mathsf{id} \times (f + i))$$

$$= \qquad \{ \ \eta \text{ natural (C.17) and } i \text{ isomorphism } \}$$

$$\mu \cdot \mathsf{B}\nabla$$
$$\cdot \mathsf{B}(\eta \cdot (\mathsf{id} \times (g + i^\circ)) \cdot (\mathsf{id} \times \iota_1) + \mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot a_p \cdot (\mathsf{id} \times \iota_2) \cdot (\mathsf{id} \times i) \cdot (\mathsf{id} \times i^\circ))$$
$$\cdot \mathsf{Bdr} \cdot a_p \cdot (\mathsf{id} \times (f + i))$$

$$= \qquad \{ \ + \text{ cancellation } \}$$

$$\mu \cdot \mathsf{B}\nabla$$
$$\cdot \mathsf{B}(\eta \cdot (\mathsf{id} \times (\iota_1 \cdot g)) + \mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot a_p \cdot (\mathsf{id} \times (f + i)) \cdot (\mathsf{id} \times \iota_2) \cdot (\mathsf{id} \times i^\circ))$$
$$\cdot \mathsf{Bdr} \cdot a_p \cdot (\mathsf{id} \times (f + i))$$

$$= \qquad \{ \text{ functors } \}$$

$$\mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta \cdot (\mathsf{id} \times \iota_1) + \mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot a_p \cdot (\mathsf{id} \times (f + i)) \cdot (\mathsf{id} \times \iota_2))$$
$$\cdot \mathsf{B}(\mathsf{id} \times g + \mathsf{id} \times i^\circ) \cdot \mathsf{Bdr} \cdot a_p \cdot (\mathsf{id} \times (f + i))$$

$$= \qquad \{ \ \mathsf{dr} \text{ natural } \}$$

$$\mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta \cdot (\mathsf{id} \times \iota_1) + \mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot a_p \cdot (\mathsf{id} \times (f + i)) \cdot (\mathsf{id} \times \iota_2)) \cdot \mathsf{Bdr}$$
$$\cdot \mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot a_p \cdot (\mathsf{id} \times (f + i))$$

$$= \qquad \{ \text{ functors } \}$$

$$\mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + \mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot a_p \cdot (\mathsf{id} \times (f + i))) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr}$$
$$\cdot \mathsf{B}(\mathsf{id} \times (g + i^\circ)) \cdot a_p \cdot (\mathsf{id} \times (f + i))$$

$$= \qquad \{ \textit{ feedback} \text{ and } \textit{wrapping} \text{ definitions } \}$$

$$a_{p[f+i,g+i^\circ]\Lsh_W}$$

□

$\boxed{\textbf{LEMMA §5.73}}$

Proof. Let us detail the proof concerning sequential composition:

$$
\begin{array}{cl}
& a_{p;q} \\
= & \quad \{ \ \textbf{;} \text{ definition and } p, q \text{ separable} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \\
& \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot ((a_{1.p} + a_{2.p}) \times \mathsf{id}) \cdot (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \text{routine: } (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{xr} = \mathsf{dl}^\circ \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \\
& \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot ((a_{1.p} + a_{2.p}) \times \mathsf{id}) \cdot \mathsf{dl}^\circ \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \quad \{ \ \mathsf{dl}^\circ \text{ natural} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \\
& \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot \mathsf{dl}^\circ \cdot (a_{1.p} \times \mathsf{id} + a_{2.p} \times \mathsf{id}) \\
& \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \quad \{ \ \text{law (C.66)} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q})) \cdot \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \\
& \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot [\mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id})] \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot \mathsf{dl}^\circ \\
& \cdot (a_{1.p} \times \mathsf{id} + a_{2.p} \times \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ fusion and } \mathsf{dl} \text{ isomorphism} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q})) \\
& \cdot [\mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}(\mathsf{id} \times \mathsf{dr}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) \cdot \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id})] \\
& \cdot (\tau_r + \tau_r) \cdot (a_{1.p} \times \mathsf{id} + a_{2.p} \times \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \quad \{ \ \mathsf{xr}, \mathsf{a} \text{ natural and } + \text{ absorption} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q})) \\
& \cdot [\mathsf{B}(\mathsf{id} \times \mathsf{dr} \cdot (\mathsf{id} \times \iota_1)), \mathsf{B}(\mathsf{id} \times \mathsf{dr} \cdot (\mathsf{id} \times \iota_2))] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (\tau_r + \tau_r) \\
& \cdot (a_{1.p} \times \mathsf{id} + a_{2.p} \times \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \quad \{ \ \text{laws (C.55) and (C.56)} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]) \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q})) \\
& \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(\mathsf{a} \cdot \mathsf{xr}) + \mathsf{B}(\mathsf{a} \cdot \mathsf{xr})) \cdot (\tau_r + \tau_r) \\
& \cdot (a_{1.p} \times \mathsf{id} + a_{2.p} \times \mathsf{id}) \cdot (\mathsf{xr} + \mathsf{xr}) \cdot \mathsf{dr} \\
= & \quad \{ \ \text{law (C.67)} \ \} \\
& \mu \cdot \mathsf{BBa}^\circ \cdot \mathsf{B}[\mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_1)), \mathsf{B}(\mathsf{id} \times (\mathsf{id} \times \iota_2))] \cdot \mathsf{B}(\tau_l + \tau_l) \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times (a_{1.q} + a_{2.q}))
\end{array}
$$

$$
\begin{array}{ll}
 & \cdot[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id})\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ +\text{ fusion and } \mathsf{a}^\circ \text{ natural}\ \} \\
 & \mu\cdot\mathsf{B}[\mathsf{B}(\mathsf{id}\times\iota_1)\cdot\mathsf{Ba}^\circ,\mathsf{B}(\mathsf{id}\times\iota_2)\cdot\mathsf{Ba}^\circ]\cdot\mathsf{B}(\tau_l+\tau_l)\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times(a_{1.q}+a_{2.q})) \\
 & \cdot[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id})\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ +\text{ absorption}\ \} \\
 & \mu\cdot\mathsf{B}[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot\mathsf{B}(\mathsf{Ba}^\circ+\mathsf{Ba}^\circ)\cdot\mathsf{B}(\tau_l+\tau_l)\cdot\mathsf{Bdr}\cdot[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)] \\
 & \cdot\mathsf{B}(\mathsf{id}\times a_{1.q})+\mathsf{B}(\mathsf{id}\times a_{2.q})\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id})\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ +\text{ fusion}\ \} \\
 & \mu\cdot\mathsf{B}[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)] \\
 & \cdot[\mathsf{B}(\mathsf{Ba}^\circ\cdot\tau_l+\mathsf{Ba}^\circ\cdot\tau_l)\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{Ba}^\circ\cdot\tau_l+\mathsf{Ba}^\circ\cdot\tau_l)\cdot\mathsf{Bdr}\cdot\mathsf{B}(\mathsf{id}\times\iota_2)] \\
 & \cdot(\mathsf{B}(\mathsf{id}\times a_{1.q})+\mathsf{B}(\mathsf{id}\times a_{2.q}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id}) \\
 & \cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ \text{laws (C.55) and (C.56)}\ \} \\
 & \mu\cdot\mathsf{B}[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot[\mathsf{B}(\mathsf{Ba}^\circ\cdot\tau_l+\mathsf{Ba}^\circ\cdot\tau_l)\cdot\mathsf{B}\iota_1,\mathsf{B}(\mathsf{Ba}^\circ\cdot\tau_l+\mathsf{Ba}^\circ\cdot\tau_l)\cdot\mathsf{B}\iota_2] \\
 & \cdot(\mathsf{B}(\mathsf{id}\times a_{1.q})+\mathsf{B}(\mathsf{id}\times a_{2.q}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id}) \\
 & \cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ +\text{ cancellation and absorption}\ \} \\
 & \mu\cdot\mathsf{B}[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot[\mathsf{B}\iota_1,\mathsf{B}\iota_2]\cdot(\mathsf{BBa}^\circ+\mathsf{BBa}^\circ)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l) \\
 & \cdot(\mathsf{B}(\mathsf{id}\times a_{1.q})+\mathsf{B}(\mathsf{id}\times a_{2.q}))\cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r) \\
 & \cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id})\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ +\text{ fusion and cancellation}\ \} \\
 & \mu\cdot[\mathsf{BB}(\mathsf{id}\times\iota_1),\mathsf{BB}(\mathsf{id}\times\iota_1)]\cdot(\mathsf{BBa}^\circ+\mathsf{BBa}^\circ)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l)\cdot(\mathsf{B}(\mathsf{id}\times a_{1.q})+\mathsf{B}(\mathsf{id}\times a_{2.q})) \\
 & \cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id})\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ +\text{ fusion and } \mu \text{ natural}\ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1)\cdot\mu,\mathsf{B}(\mathsf{id}\times\iota_1)\cdot\mu]\cdot(\mathsf{BBa}^\circ+\mathsf{BBa}^\circ)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l)\cdot(\mathsf{B}(\mathsf{id}\times a_{1.q})+\mathsf{B}(\mathsf{id}\times a_{2.q})) \\
 & \cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id})\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr} \\
= & \qquad \{\ +\text{ absorption}\ \} \\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mu+\mu)\cdot(\mathsf{BBa}^\circ+\mathsf{BBa}^\circ)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l)\cdot\mathsf{B}(\mathsf{id}\times a_{1.q})+\mathsf{B}(\mathsf{id}\times a_{2.q}) \\
 & \cdot(\mathsf{B}(\mathsf{a}\cdot\mathsf{xr})+\mathsf{B}(\mathsf{a}\cdot\mathsf{xr}))\cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times\mathsf{id}+a_{2.p}\times\mathsf{id})\cdot(\mathsf{xr}+\mathsf{xr})\cdot\mathsf{dr}
\end{array}
$$

$$= \quad \{ \ ; \text{ definition } \}$$

$$[\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p;1.q}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p;2.q}] \cdot \mathsf{dr}$$

The other facts in the first group are trivial. Just notice that, in the *wrapping* case, separability propagates only when the wrapping functions are sums. In the second group of compositions, we prove the *parallel* case below. The others follow similar arguments. Thus,

$$a_{(p \boxtimes r)[\mathsf{dl}^\circ, \mathsf{dl}]}$$

$$= \quad \{ \ p \text{ separable, } \boxtimes \text{ and wrapping definitions } \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot \mathsf{Bm} \cdot \delta_l \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot ((a_{1.p} + a_{2.p}) \times a_r) \cdot (\mathsf{dr} \times \mathsf{id})$$
$$\cdot \mathsf{m} \cdot (\mathsf{id} \times \mathsf{dl}^\circ)$$

$$= \quad \{ \ \text{routine: } (\mathsf{dr} \times \mathsf{id}) \cdot \mathsf{m} \cdot (\mathsf{id} \times \mathsf{dl}^\circ) = \mathsf{dl}^\circ \cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr} \ \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot \mathsf{Bm} \cdot \delta_l \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot ((a_{1.p} + a_{2.p}) \times a_r) \cdot \mathsf{dl}^\circ$$
$$\cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr}$$

$$= \quad \{ \ \mathsf{dl}^\circ \text{ natural } \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot \mathsf{Bm} \cdot \delta_l \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot \mathsf{dl}^\circ \cdot (a_{1.p} \times a_r + a_{2.p} \times a_r)$$
$$\cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr}$$

$$= \quad \{ \ \delta_l \text{ definition } \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot \mathsf{Bm} \cdot \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot ([\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \times \mathsf{id}) \cdot \mathsf{dl}^\circ \cdot (a_{1.p} \times a_r + a_{2.p} \times a_r)$$
$$\cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr}$$

$$= \quad \{ \ \text{law (C.66)} \ \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot \mathsf{Bm} \cdot \mu \cdot \mathsf{B}\tau_l \cdot [\mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id})] \cdot (\tau_r + \tau_r) \cdot \mathsf{dl} \cdot \mathsf{dl}^\circ$$
$$\cdot (a_{1.p} \times a_r + a_{2.p} \times a_r) \cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr}$$

$$= \quad \{ \ \mathsf{dl}^\circ \text{ isomorphism, } + \text{ fusion } \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot \mathsf{Bm} \cdot \mu \cdot [\mathsf{B}\tau_l \cdot \mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{B}\tau_l \cdot \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id})]$$
$$\cdot (\tau_r + \tau_r) \cdot (a_{1.p} \times a_r + a_{2.p} \times a_r) \cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr}$$

$$= \quad \{ \ \tau_l \text{ natural (C.6), } + \text{ absorption } \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot \mathsf{Bm} \cdot \mu \cdot [\mathsf{BB}((\mathsf{id} \times \iota_1) \times \mathsf{id}), \mathsf{BB}((\mathsf{id} \times \iota_2) \times \mathsf{id})] \cdot (\mathsf{B}\tau_l + \mathsf{B}\tau_l) \cdot (\tau_r + \tau_r)$$
$$\cdot (a_{1.p} \times a_r + a_{2.p} \times a_r) \cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr}$$

$$= \quad \{ \ \mu \text{ natural (C.16), } + \text{ fusion } \}$$

$$\mathsf{B}(\mathsf{id} \times \mathsf{dl}) \cdot [\mathsf{Bm} \cdot \mathsf{B}((\mathsf{id} \times \iota_1) \times \mathsf{id}) \cdot \mu, \mathsf{Bm} \cdot \mathsf{B}((\mathsf{id} \times \iota_2) \times \mathsf{id}) \cdot \mu] \cdot (\mathsf{B}\tau_l + \mathsf{B}\tau_l)$$
$$\cdot (\tau_r + \tau_r) \cdot (a_{1.p} \times a_r + a_{2.p} \times a_r) \cdot (\mathsf{m} + \mathsf{m}) \cdot \mathsf{dr}$$

$$= \quad \{ \ + \text{ absorption } \}$$

$$\begin{array}{ll}
 & \mathsf{B}(\mathsf{id}\times\mathsf{dl})\cdot[\mathsf{Bm}\cdot\mathsf{B}((\mathsf{id}\times\iota_1)\times\mathsf{id}),\mathsf{Bm}\cdot\mathsf{B}((\mathsf{id}\times\iota_2)\times\mathsf{id})]\cdot(\mu+\mu)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l)\\
 & \cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times a_r+a_{2.p}\times a_r)\cdot(\mathsf{m}+\mathsf{m})\cdot\mathsf{dr}\\
= & \quad\{\ \text{routine: } \mathsf{m}\cdot((\mathsf{id}\times\iota_1)\times\mathsf{id})=(\mathsf{id}\times\mathsf{dl}^\circ)\cdot(\mathsf{id}\times\iota_1)\cdot\mathsf{m}\ \}\\
 & \mathsf{B}(\mathsf{id}\times\mathsf{dl})\cdot[\mathsf{B}(\mathsf{id}\times\mathsf{dl}^\circ)\cdot\mathsf{B}(\mathsf{id}\times\iota_1)\cdot\mathsf{Bm},\mathsf{B}(\mathsf{id}\times\mathsf{dl}^\circ)\cdot\mathsf{B}(\mathsf{id}\times\iota_2)\cdot\mathsf{Bm}]\\
 & \cdot(\mu+\mu)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l)\cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times a_r+a_{2.p}\times a_r)\cdot\mathsf{dl}^\circ\cdot(\mathsf{m}+\mathsf{m})\cdot\mathsf{dr}\\
= & \quad\{\ +\text{ absorption and fusion }\}\\
 & \mathsf{B}(\mathsf{id}\times\mathsf{dl})\cdot\mathsf{B}(\mathsf{id}\times\mathsf{dl}^\circ)\cdot[\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(\mathsf{Bm}+\mathsf{Bm})\cdot(\mu+\mu)\cdot(\mathsf{B}\tau_l+\mathsf{B}\tau_l)\\
 & \cdot(\tau_r+\tau_r)\cdot(a_{1.p}\times a_r+a_{2.p}\times a_r)\cdot\mathsf{dl}^\circ\cdot(\mathsf{m}+\mathsf{m})\cdot\mathsf{dr}\\
= & \quad\{\ \mathsf{dl}\text{ isomorphism and }\boxtimes\text{ definition }\}\\
 & [\mathsf{B}(\mathsf{id}\times\iota_1),\mathsf{B}(\mathsf{id}\times\iota_2)]\cdot(a_{1.p\boxtimes r}+a_{2.p\boxtimes r})\cdot\mathsf{dr}
\end{array}$$

□

LEMMA §5.77

Proof. Note that both components have type $I \longrightarrow O + Z$. Then, by law (5.10),

$$\ulcorner \iota_1 \urcorner \,;\, (p[\mathsf{id}, \mathsf{s}_+]) \Lsh_Z \;\sim\; ((p[\mathsf{id}, \mathsf{s}_+]) \Lsh_Z)[\iota_1, \mathsf{id}]$$

Thus,

$$
\begin{array}{ll}
 & a_{((p[\mathsf{id},\mathsf{s}_+])\Lsh_Z)[\iota_1,\mathsf{id}]} \\
= & \quad \{ \ \textit{partial feedback} \text{ definition, } p \text{ separable} \ \} \\
 & \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \\
 & \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_1) \\
= & \quad \{ \ \text{law (C.55)} \ \} \\
 & \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \\
 & \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \iota_1 \\
= & \quad \{ \ + \text{ cancellation} \ \} \\
 & \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p} \\
= & \quad \{ \ \mathsf{dr} \text{ natural} \ \} \\
 & \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bs}_+ \cdot \mathsf{Bdr} \cdot \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p} \\
= & \quad \{ \ \text{law (C.55)} \ \} \\
 & \mu \cdot \mathsf{B}\nabla \cdot \mathsf{B}(\eta + \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p) \cdot \mathsf{B}(\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{Bs}_+ \cdot \mathsf{B}\iota_1 \cdot a_{1.p} \\
= & \quad \{ \ \nabla \text{ definition and } + \text{ absorption} \ \} \\
 & \mu \cdot \mathsf{B}[\eta \cdot (\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p \cdot (\mathsf{id} \times \iota_2)] \cdot \mathsf{Bs}_+ \cdot \mathsf{B}\iota_1 \cdot a_{1.p} \\
= & \quad \{ \ + \text{ fusion application: } [f, g] \cdot \mathsf{s}_+ = [g, f] \ \} \\
 & \mu \cdot \mathsf{B}[\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p \cdot (\mathsf{id} \times \iota_2), \eta \cdot (\mathsf{id} \times \iota_1)] \cdot \mathsf{B}\iota_1 \cdot a_{1.p} \\
= & \quad \{ \ + \text{ cancellation} \ \} \\
 & \mu \cdot \mathsf{B}(\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot a_p \cdot (\mathsf{id} \times \iota_2)) \cdot a_{1.p} \\
= & \quad \{ \ p \text{ separable} \ \} \\
 & \mu \cdot \mathsf{B}(\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_2)) \cdot a_{1.p} \\
= & \quad \{ \ \text{law (C.54)} \ \} \\
 & \mu \cdot \mathsf{B}(\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \iota_2) \cdot a_{1.p} \\
= & \quad \{ \ + \text{ cancellation} \ \} \\
 & \mu \cdot \mathsf{B}(\mathsf{B}(\mathsf{id} \times \mathsf{s}_+) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}) \cdot a_{1.p} \\
= & \quad \{ \ \text{routine: } \mathsf{s}_+ \cdot \iota_2 = \iota_1 \ \}
\end{array}
$$

$$
\begin{array}{ll}
 & \mu \cdot \mathsf{B}(\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{2.p}) \cdot a_{1.p} \\
= & \qquad \{ \ \mu \text{ natural (C.16)} \ \} \\
 & \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mu \cdot \mathsf{B}a_{2.p} \cdot a_{1.p} \\
= & \qquad \{ \ \bullet \text{ definition} \ \} \\
 & \mathsf{B}(\mathsf{id} \times \iota_1) \cdot (a_{2.p} \bullet a_{1.p}) \\
= & \qquad \{ \ \textit{hook} \text{ definition} \ \} \\
 & a_{(\left(p\right)_Z)[\mathsf{id},\iota_1]}
\end{array}
$$

which, again by law (5.10), is bisimilar to $\left(p\right)_Z \,;\, \ulcorner \iota_1 \urcorner$.

□

3. Proofs for Chapter 6

LEMMA §6.5

Proof. Let us check the *action* part of equations (6.3) and (6.4). For the first one, consider:

$$
\begin{array}{cl}
& \mathsf{B}\pi_2 \cdot a_{\mathsf{copy}_I;p} \\
= & \qquad \{ \text{ ; and } \mathsf{copy}_I \text{ definitions } \} \\
& \mathsf{B}\pi_2 \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \mu \text{ natural (C.16) } \} \\
& \mu \cdot \mathsf{BB}\pi_2 \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ law (C.13) } \} \\
& \mu \cdot \mathsf{B}\pi_2 \cdot \mathsf{B}(\mathsf{id} \times a_p) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \times \text{ cancellation } \} \\
& \mu \cdot \mathsf{B}a_p \cdot \mathsf{B}\pi_2 \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ routine: } \pi_2 \cdot \mathsf{a} = \pi_2 \times \mathsf{id} \ \} \\
& \mu \cdot \mathsf{B}a_p \cdot \mathsf{B}(\pi_2 \times \mathsf{id}) \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \times \text{ absorption } \} \\
& \mu \cdot \mathsf{B}a_p \cdot \mathsf{B}\langle \pi_2, \pi_1 \rangle \cdot \tau_r \cdot ((\eta \cdot \pi_2) \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \eta \text{ strong (C.18) } \} \\
& \mu \cdot \mathsf{B}a_p \cdot \mathsf{B}\langle \pi_2, \pi_1 \rangle \cdot \eta \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ \eta \text{ natural (C.17) } \} \\
& \mu \cdot \eta \cdot a_p \cdot \langle \pi_2, \pi_1 \rangle \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ law (C.14) } \} \\
& a_p \cdot \langle \pi_2, \pi_1 \rangle \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \text{ routine: } \langle \pi_2, \pi_1 \rangle \cdot (\pi_2 \times \mathsf{id}) = (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \ \} \\
& a_p \cdot (\pi_2 \times \mathsf{id}) \cdot \mathsf{xr} \cdot \mathsf{xr} \\
= & \qquad \{ \ \mathsf{xr}^\circ = \mathsf{xr} \ \} \\
& a_p \cdot (\pi_2 \times \mathsf{id})
\end{array}
$$

Now, for equation (6.4),

$$
\begin{array}{ll}
& \mathsf{B}\pi_1 \cdot a_{p;\mathsf{copy}_O} \\
= & \quad \{ \text{ ; and } \mathsf{copy}_O \text{ definitions } \} \\
& \mathsf{B}\pi_1 \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times (\eta \cdot \pi_2)) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ law (C.20) } \} \\
& \mathsf{B}\pi_1 \cdot \mu \cdot \mathsf{B}\eta \cdot \mathsf{B}(\mathsf{id} \times \pi_2) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ law (C.14) } \} \\
& \mathsf{B}\pi_1 \cdot \mathsf{B}(\mathsf{id} \times \pi_2) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ routine: } \pi_1 \cdot (\mathsf{id} \times \pi_2) \cdot \mathsf{a} = \pi_1 \cdot \pi_1 \ \} \\
& \mathsf{B}\pi_1 \cdot \mathsf{B}\pi_1 \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \times \text{ cancellation } \} \\
& \mathsf{B}\pi_1 \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ law (C.12) } \} \\
& \pi_1 \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \ \times \text{ cancellation } \} \\
& a_p \cdot \pi_1 \cdot \mathsf{xr} \\
= & \quad \{ \text{ routine: } \pi_1 \cdot \mathsf{xr} = \pi_1 \times \mathsf{id} \ \} \\
& a_p \cdot (\pi_1 \times \mathsf{id})
\end{array}
$$

□

LEMMA §6.12

Proof. Let $h : p \longrightarrow p'$ and $k : q \longrightarrow q'$. It remains to be shown that

$$\mathsf{B}(h \times k) \cdot a_{p;q} \;=\; a_{p';q'} \cdot ((h \times k) \times \mathsf{id}) \qquad \text{(D.8)}$$

and

$$\mathsf{B}o_{p;q} \cdot a_{p;q} \;=\; \mathsf{B}o_{p';q'} \cdot a_{p';q'} \cdot ((h \times k) \times \mathsf{id}) \qquad \text{(D.9)}$$

Equation (D.8) is checked through the following calculation:

$$\begin{array}{ll}
& a_{p';q'} \cdot ((h \times k) \times \mathsf{id}) \\
= & \quad \{ \text{ ; definition } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_{p'} \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_{p'} \times \mathsf{id}) \cdot \mathsf{xr} \cdot ((h \times k) \times \mathsf{id}) \\
= & \quad \{ \text{ } \mathsf{xr} \text{ natural } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_{p'} \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_{p'} \times \mathsf{id}) \cdot ((h \times \mathsf{id}) \times k) \cdot \mathsf{xr} \\
= & \quad \{ \text{ assumption: } \mathsf{B}h \cdot a_p = a_{p'} \cdot (h \times \mathsf{id}) \text{ } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_{p'} \cdot \pi_1 \rangle \cdot \tau_r \cdot (\mathsf{B}h \times k) \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ } \tau_r \text{ natural (C.5) } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_{p'} \cdot \pi_1 \rangle \cdot \mathsf{B}(h \times k) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ } \times \text{ fusion } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle h \times k, o_{p'} \cdot \pi_1 \cdot (h \times k) \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ } \times \text{ cancellation } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle h \times k, o_{p'} \cdot h \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ by assumption } h \text{ is a next comorphism which implies } o_p = o_{p'} \cdot h \text{ (lemma §6.7) } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle h \times k, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ } \times \text{ absorption } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{Ba} \cdot \mathsf{B}((h \times k) \times \mathsf{id}) \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ } \mathsf{a} \text{ natural } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_{q'}) \cdot \mathsf{B}(h \times (k \times \mathsf{id})) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ assumption: } \mathsf{B}k \cdot a_q = a_{q'} \cdot (k \times \mathsf{id}) \text{ } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(h \times \mathsf{B}k) \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(h \times (k \times \mathsf{id})) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \\
& \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{ \text{ } \tau_l \text{ natural (C.6) } \}
\end{array}$$

$$
\begin{array}{ll}
 & \mu \cdot \mathsf{BB}(h \times k) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(h \times (k \times \mathsf{id})) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \\
 & \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{\ \mu \text{ natural (C.16)}\ \} \\
 & \mathsf{B}(h \times k) \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(h \times (k \times \mathsf{id})) \cdot \mathsf{Ba} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \\
 & \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{\ \boldsymbol{;} \text{ definition}\ \} \\
 & \mathsf{B}(h \times k) \cdot a_{p;q}
\end{array}
$$

For equation (D.9) consider:

$$
\begin{array}{ll}
 & \mathsf{B}o_{p';q'} \cdot a_{p';q'} \cdot ((h \times k) \times \mathsf{id}) \\
= & \qquad \{\ \text{just proved}\ \} \\
 & \mathsf{B}o_{p';q'} \cdot \mathsf{B}(h \times k) \cdot a_{p;q} \\
= & \qquad \{\ \boldsymbol{;} \text{ definition}\ \} \\
 & \mathsf{B}o_{q'} \cdot \mathsf{B}\pi_2 \cdot \mathsf{B}(h \times k) \cdot a_{p;q} \\
= & \qquad \{\ \times \text{ cancellation}\ \} \\
 & \mathsf{B}o_{q'} \cdot \mathsf{B}k \cdot \mathsf{B}\pi_2 \cdot a_{p;q} \\
= & \qquad \{\ \text{by assumption } k \text{ is a next comorphism, which implies } o_q = o_{q'} \cdot h \text{ (lemma §6.7)}\ \} \\
 & \mathsf{B}o_q \cdot \mathsf{B}\pi_2 \cdot a_{p;q} \\
= & \qquad \{\ \boldsymbol{;} \text{ definition}\ \} \\
 & \mathsf{B}o_{p;p} \cdot a_{p;q}
\end{array}
$$

□

LEMMA §6.23

Proof. All these laws are, in fact, strict equalities. For equation (6.23) observe first that $o_{\ulcorner f \urcorner^{\Lsh}} = o_{\ulcorner f \urcorner}$ and this, in turn, equals $o_{\ulcorner f . f \urcorner}$. For the action part, reason as follows

$$\begin{array}{ll}
 & a_{\ulcorner f \cdot f \urcorner} \\
= & \qquad \{ \textit{ feedback } \text{definition } \} \\
 & \mu \cdot \mathsf{B} a_{\ulcorner f \urcorner} \cdot \mathsf{B}\langle \mathsf{id}, o_{\ulcorner f \urcorner} \rangle \cdot a_{\ulcorner f \urcorner} \\
= & \qquad \{ \textit{ function lifting } \text{definition } \} \\
 & \mu \cdot \mathsf{B}\eta \cdot \mathsf{B}f \cdot \mathsf{B}\pi_2 \cdot \mathsf{B}\langle \mathsf{id}, \mathsf{id} \rangle \cdot \eta \cdot f \cdot \pi_2 \\
= & \qquad \{ \text{ law (C.14) } \} \\
 & \mathsf{B}f \cdot \mathsf{B}\pi_2 \cdot \mathsf{B}\langle \mathsf{id}, \mathsf{id} \rangle \cdot \eta \cdot f \cdot \pi_2 \\
= & \qquad \{ \times \text{ cancellation } \} \\
 & \mathsf{B}f \cdot \eta \cdot f \cdot \pi_2 \\
= & \qquad \{ \eta \text{ natural (C.17) } \} \\
 & \eta \cdot f \cdot f \cdot \pi_2 \\
= & \qquad \{ \textit{ function lifting } \text{definition } \} \\
 & a_{\ulcorner f \cdot f \urcorner}
\end{array}$$

Consider, now, equation (6.24). By definition of *wrapping* one gets $o_{r^{\Lsh}[i,i^\circ]} = i^\circ \cdot o_{r^{\Lsh}}$, which equals $i^\circ \cdot o_r$. On the other hand, $o_{r[i,i^\circ]^{\Lsh}} = o_{r[i,i^\circ]} = i^\circ \cdot o_r$. For the action part, consider

$$\begin{array}{ll}
 & a_{r[i,i^\circ]^{\Lsh}} \\
= & \qquad \{ \textit{ feedback } \text{and } \textit{wrapping } \text{definitions } \} \\
 & \mu \cdot \mathsf{B}a_r \cdot \mathsf{B}(\mathsf{id} \times i) \cdot \mathsf{B}\langle \mathsf{id}, i^\circ \cdot o_r \rangle \cdot a_r \cdot (\mathsf{id} \times i) \\
= & \qquad \{ \times \text{ absorption } \} \\
 & \mu \cdot \mathsf{B}a_r \cdot \mathsf{B}\langle \mathsf{id}, i \cdot i^\circ \cdot o_r \rangle \cdot a_r \cdot (\mathsf{id} \times i) \\
= & \qquad \{ i \text{ isomorphism } \} \\
 & \mu \cdot \mathsf{B}a_r \cdot \mathsf{B}\langle \mathsf{id}, o_r \rangle \cdot a_r \cdot (\mathsf{id} \times i) \\
= & \qquad \{ \textit{ feedback } \text{and } \textit{wrapping } \text{definitions } \} \\
 & a_{r^{\Lsh}[i,i^\circ]}
\end{array}$$

For the remaining laws, observe that the attribute parts of both sides of each equation coincide trivially. The action parts are proved as in §5.57.

□

LEMMA §6.27

Proof. We shall consider in detail equation (6.28), which is a strict equality. Note that (6.29) is simply a corollary of this, because $(p \boxast q)[\iota_1, \mathsf{id}] \sim p \boxplus q$. Thus, for the attribute part reason:

$$\begin{array}{ll}
 & o_{(p\boxplus q)[\mathsf{id},\mathsf{s}]\Lsh_Z[\iota_1,\pi_1]} \\
= & \qquad \{\ \textit{wrapping}, \textit{partial feedback} \text{ and } \boxplus \text{ definitions}\ \} \\
 & \mathsf{B}\pi_1 \cdot \mathsf{Bs} \cdot (o_p \times o_q) \\
= & \qquad \{\ \mathsf{s} \text{ natural}\ \} \\
 & \mathsf{B}\pi_1 \cdot (o_q \times o_p) \cdot \mathsf{Bs} \\
= & \qquad \{\ \times \text{ cancellation}\ \} \\
 & \mathsf{B}\pi_1 \cdot o_q \cdot \pi_1 \cdot \mathsf{Bs} \\
= & \qquad \{\ \mathsf{s} \text{ definition and } \times \text{ cancellation}\ \} \\
 & \mathsf{B}\pi_1 \cdot o_q \cdot \pi_2 \\
= & \qquad \{\ ; \text{ definition}\ \} \\
 & o_{p;q}
\end{array}$$

For the action part, consider

$$\begin{array}{ll}
 & a_{(p\boxplus q)[\mathsf{id},\mathsf{s}]\Lsh_Z[\iota_1,\pi_1]} \\
= & \qquad \{\ \textit{wrapping} \text{ definition}\ \} \\
 & a_{(p\boxplus q)[\mathsf{id},\mathsf{s}]\Lsh_Z} \cdot (\mathsf{id} \times \iota_1) \\
= & \qquad \{\ \textit{partial feedback} \text{ and } \textit{wrapping} \text{ definitions}\ \} \\
 & \mu \cdot \mathsf{B}a_{p\boxplus q} \cdot \mathsf{B}\langle \mathsf{id}, \pi_2 \cdot \mathsf{s} \cdot o_{p\boxplus q}\rangle \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{p\boxplus q} \cdot (\mathsf{id} \times \iota_1) \\
= & \qquad \{\ \mathsf{s} \text{ definition and } \times \text{ cancellation}\ \} \\
 & \mu \cdot \mathsf{B}a_{p\boxplus q} \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \cdot o_{p\boxplus q}\rangle \cdot a_{p\boxplus q} \cdot (\mathsf{id} \times \iota_1) \\
= & \qquad \{\ \boxplus \text{ definition}\ \} \\
 & \mu \cdot \mathsf{B}(\nabla \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{B}\langle \mathsf{id}, \pi_1 \cdot o_{p\boxplus q}\rangle \\
 & \cdot \nabla \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_1) \\
= & \qquad \{\ \times \text{ cancellation}\ \} \\
 & \mu \cdot \mathsf{B}(\nabla \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr}) \cdot \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1\rangle \\
 & \cdot \nabla \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \mathsf{dr} \cdot (\mathsf{id} \times \iota_1) \\
= & \qquad \{\ \text{laws (C.55) and (C.56)}\ \} \\
 & \mu \cdot \mathsf{B}(\nabla \cdot (\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a})) \cdot \mathsf{B}\iota_2 \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1\rangle \cdot \nabla
\end{array}$$

$$\begin{aligned}
& \cdot(\tau_r + \tau_l) \cdot (a_p \times \mathsf{id} + \mathsf{id} \times a_q) \cdot (\mathsf{xr} + \mathsf{a}) \cdot \iota_1 \\
= & \qquad \{ \ \nabla \text{ definition and } \times \text{ absorption } \} \\
& \mu \cdot \mathsf{B}[\tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr}, \tau_l \cdot (\mathsf{id} \times a_q) \cdot \mathsf{a}] \cdot \mathsf{B}\iota_2 \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \\
& \cdot[\tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr}, \tau_l \cdot (\mathsf{id} \times a_q) \cdot \mathsf{a}] \cdot \iota_1 \\
= & \qquad \{ \ + \text{ cancellation } \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot (\mathsf{id} \times a_q) \cdot \mathsf{a} \cdot \mathsf{B}\langle \mathsf{id}, o_p \cdot \pi_1 \rangle \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \qquad \{ \ ; \text{ definition } \} \\
& a_{p;q}
\end{aligned}$$

Finally, note that $\gamma_{(p \boxplus q)[\mathsf{id},\mathsf{s}] \Lsh_Z [\iota_1,\pi_1]} = \wedge \cdot (\gamma_p \times \gamma_q) = \gamma_{p;q}$. by definition of both *wrapping* and *partial feedback*,

□

$\boxed{\textsc{Lemma}\ \S 6.30}$

Proof. Consider equation (6.30) first, which is a strict equality. Thus,

$$\begin{array}{ll}
 & o_{\delta(p[f,g])} \\
= & \quad \{ \textit{ delay} \text{ definition } \} \\
 & o_{p[f,g]} \cdot \pi_1 \\
= & \quad \{ \textit{ wrapping} \text{ definition } \} \\
 & g \cdot o_p \cdot \pi_1 \\
= & \quad \{ \textit{ delay} \text{ definition } \} \\
 & g \cdot o_{\delta p} \\
= & \quad \{ \textit{ wrapping} \text{ definition } \} \\
 & o_{(\delta p)[f,g]}
\end{array}$$

For the action part compute

$$\begin{array}{ll}
 & a_{\delta(p[f,g])} \\
= & \quad \{ \textit{ delay} \text{ definition } \} \\
 & \tau_l \cdot (\mathsf{id} \times a_{p[f,g]}) \cdot \mathsf{a} \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= & \quad \{ \textit{ wrapping} \text{ definition } \} \\
 & \tau_l \cdot (\mathsf{id} \times a_p) \cdot (\mathsf{id} \times (\mathsf{id} \times f)) \cdot \mathsf{a} \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= & \quad \{ \text{ } \mathsf{a} \text{ natural } \} \\
 & \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot (\mathsf{id} \times f) \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= & \quad \{ \text{ functors } \} \\
 & \tau_l \cdot (\mathsf{id} \times a_p) \cdot \mathsf{a} \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \cdot (\mathsf{id} \times f) \\
= & \quad \{ \textit{ wrapping} \text{ and } \textit{delay} \text{ definitions } \} \\
 & a_{(\delta p)[f,g]}
\end{array}$$

Bisimilarity in the remaining three equations is witnessed by the exchange natural isomorphism $\mathsf{m} : U_p \times U_p \times (U_q \times U_q) \longrightarrow U_p \times U_q \times (U_p \times U_q)$ as a comorphism from left to the righthand side. We shall detail the proof of equation (6.31); the remaining two are similar resorting to the combinators definitions and properties of m collected in appendix C. Notice, in particular, the observers part for all equations are established by the same calculation. Thus,

$$\begin{array}{ll}
 & o_{\delta p \boxtimes \delta q} \cdot \mathsf{m} \\
= & \quad \{ \text{ } \boxtimes \text{ and } \textit{delay} \text{ definitions } \} \\
 & (o_p \cdot \pi_1 \times o_q \cdot \pi_1) \cdot \mathsf{m}
\end{array}$$

$$\begin{aligned}
= &\quad \{\ \times \text{ functor}\ \} \\
& (o_p \times o_q) \cdot (\pi_1 \times \pi_1) \cdot \mathsf{m} \\
= &\quad \{\ \text{routine: } (\pi_1 \times \pi_1) \cdot \mathsf{m} = \pi_1\ \} \\
& (o_p \times o_q) \cdot \pi_1 \\
= &\quad \{\ \textit{delay} \text{ definition}\ \} \\
& o_{\delta(p \boxtimes q)}
\end{aligned}$$

For the action part compute

$$\begin{aligned}
& \mathsf{Bm} \cdot a_{\delta(p \boxtimes q)} \\
= &\quad \{\ \textit{delay} \text{ definition}\ \} \\
& \mathsf{Bm} \cdot \tau_l \cdot (\mathsf{id} \times a_{p \boxplus q}) \cdot \mathsf{a} \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= &\quad \{\ \boxtimes \text{ definition}\ \} \\
& \mathsf{Bm} \cdot \tau_l \cdot (\mathsf{id} \times \delta_l) \cdot (\mathsf{id} \times (a_p \times a_q)) \cdot (\mathsf{id} \times \mathsf{m}) \cdot \mathsf{a} \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= &\quad \{\ \text{law (C.81)}\ \} \\
& \delta_l \cdot (\tau_l \times \tau_l) \cdot \mathsf{m} \cdot (\mathsf{id} \times (a_p \times a_q)) \cdot (\mathsf{id} \times \mathsf{m}) \cdot \mathsf{a} \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= &\quad \{\ \mathsf{m} \text{ natural}\ \} \\
& \delta_l \cdot (\tau_l \times \tau_l) \cdot ((\mathsf{id} \times a_p) \times (\mathsf{id} \times a_q)) \cdot \mathsf{m} \cdot (\mathsf{id} \times \mathsf{m}) \cdot \mathsf{a} \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= &\quad \{\ \text{routine: } \mathsf{m} \cdot (\mathsf{id} \times \mathsf{m}) \cdot \mathsf{a} = (\mathsf{a} \times \mathsf{a}) \cdot (\mathsf{m} \times \mathsf{id}) \cdot \mathsf{m}\ \} \\
& \delta_l \cdot (\tau_l \times \tau_l) \cdot ((\mathsf{id} \times a_p) \times (\mathsf{id} \times a_q)) \cdot (\mathsf{a} \times \mathsf{a}) \cdot \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) \cdot (\triangle \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id}) \\
= &\quad \{\ \text{routine: } \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) \cdot (\triangle \times \mathsf{id}) = ((\triangle \times \mathsf{id}) \times (\triangle \times \mathsf{id})) \cdot \mathsf{m}\ \} \\
& \delta_l \cdot (\tau_l \times \tau_l) \cdot ((\mathsf{id} \times a_p) \times (\mathsf{id} \times a_q)) \cdot (\mathsf{a} \times \mathsf{a}) \cdot ((\triangle \times \mathsf{id}) \times (\triangle \times \mathsf{id})) \cdot \mathsf{m} \\
& \cdot (\pi_2 \times \mathsf{id}) \\
= &\quad \{\ \text{routine: } \mathsf{m} \cdot (\pi_2 \times \mathsf{id}) = ((\pi_2 \times \mathsf{id}) \times (\pi_2 \times \mathsf{id})) \cdot \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id})\ \} \\
& \delta_l \cdot (\tau_l \times \tau_l) \cdot ((\mathsf{id} \times a_p) \times (\mathsf{id} \times a_q)) \cdot (\mathsf{a} \times \mathsf{a}) \cdot ((\triangle \times \mathsf{id}) \times (\triangle \times \mathsf{id})) \\
& \cdot ((\pi_2 \times \mathsf{id}) \times (\pi_2 \times \mathsf{id})) \cdot \mathsf{m} \cdot (\mathsf{m} \times \mathsf{id}) \\
= &\quad \{\ \times \text{ functor and } \boxtimes \text{ definition}\ \} \\
& a_{\delta p \boxtimes \delta q} \cdot (\mathsf{m} \times \mathsf{id})
\end{aligned}$$

Finally, notice the seed predicate is trivially verified in all cases.

□

$\boxed{\text{LEMMA §6.31}}$

Proof. In order to prove equation (6.34) we check that $\pi_2 \times \mathsf{id} : U_p \times U_p \times U_q \longrightarrow U_p \times U_q$ is a comorphism from $\delta p \,;\, q$ to $\big((p \boxtimes q)[\mathsf{id}, \mathsf{s}]\big)_Z$. The attribute's part is rather simple:

$$
\begin{array}{cl}
 & o_{\left((p\boxtimes q)[\mathsf{id},\mathsf{s}]\right)_Z} \cdot (\pi_2 \times \mathsf{id}) \\
= & \qquad \{\ \textit{hook}, \textit{delay} \text{ and } \boxtimes \text{ definitions}\ \} \\
 & \pi_2 \cdot \mathsf{s} \cdot (o_p \times o_q) \cdot (\pi_2 \times \mathsf{id}) \\
= & \qquad \{\ \mathsf{s} \text{ definition and } \times \text{ cancellation}\ \} \\
 & o_q \cdot \pi_2 \cdot (\pi_2 \times \mathsf{id}) \\
= & \qquad \{\ \times \text{ cancellation}\ \} \\
 & o_q \cdot \pi_2 \\
= & \qquad \{\ ; \text{ definition}\ \} \\
 & o_{\delta p;q}
\end{array}
$$

For the action part we have to show that

$$\mathsf{B}\delta p \,;\, q \cdot a_{\delta p;q} \;=\; a_{\left((p\boxtimes q)[\mathsf{id},\mathsf{s}]\right)_Z} \cdot ((\pi_2 \times \mathsf{id}) \times \mathsf{id})$$

The proof is structured according to the following diagram in which the comorphism condition is decomposed in a number of small steps:

$$\begin{tikzcd}
U_p \times U_p \times U_q \times I \arrow[rr, "\pi_2 \times \mathsf{id} \times \mathsf{id}"] \arrow[d, "\psi_1"'] \arrow[dd, bend right=60, "a_{\delta p;q}"'] & & U_p \times U_q \times I \arrow[d, "\psi_2"] \\
Z \times (U_p \times (U_p \times I)) \times U_q \arrow[r, "\mathsf{id} \times \pi_2 \times \mathsf{id}"] \arrow[d, "\phi_1"'] & Z \times (U_p \times I) \times U_q \arrow[r, "\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}"] \arrow[d, "\phi_2"] & (U_p \times I) \times (U_q \times Z) \arrow[d, "\phi_3"] \\
\mathsf{B}(U_p \times U_p \times U_q) \arrow[r, "\mathsf{B}(\pi_2 \times \mathsf{id})"] & \mathsf{B}(U_p \times U_q) \arrow[r, equal] & \mathsf{B}(U_p \times U_q)
\end{tikzcd}$$

where

$$
\begin{array}{lcl}
\psi_1 & = & (\mathsf{a} \times \mathsf{id}) \cdot (o_p \times \mathsf{id} \times \mathsf{id} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr} \\
\psi_2 & = & \mathsf{m} \cdot \langle \pi_2, \langle \pi_1, \pi_2 \cdot o_q \cdot \pi_2 \cdot \pi_1 \rangle \rangle \\
\phi_1 & = & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id}) \\
\phi_2 & = & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
\phi_3 & = & \delta_l \cdot (a_p \times a_q)
\end{array}
$$

Let us denote the topmost diagram by (a) and the left (respectively, right) lower diagram by (b) (respectively, (c)). The intuition is that diagram (a) caters for, in the pipeline expression, the Z parameter being computed earlier. Diagram (b) deals with the effect of π_2 whereas diagram (c) relates an expression in which p and q actions are computed sequentially with another in which this computation is simultaneous. Clearly, such is the case in a $\boxtimes$ expression. First notice that

$$
\begin{array}{ll}
 & a_{\left((p \boxtimes q)[\mathsf{id},\mathsf{s}]\right)_Z} \\
= & \quad \{\ \textit{hook} \text{ definition } \} \\
 & a_{(p \boxtimes q)[\mathsf{id},\mathsf{s}]} \cdot \langle \pi_2, \langle \pi_1, \pi_2 \cdot o_{(p \boxtimes q)[\mathsf{id},\mathsf{s}]} \cdot \pi_1 \rangle \rangle \\
= & \quad \{\ \boxtimes \text{ and } \textit{wrapping} \text{ definitions } \} \\
 & \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot \langle \pi_2, \langle \pi_1, \pi_2 \cdot o_q \cdot \pi_2 \cdot \pi_1 \rangle \rangle \\
= & \quad \{\ \psi_2 \text{ and } \phi_3 \text{ definitions above } \} \\
 & \phi_3 \cdot \psi_2
\end{array}
$$

and that the commutativity of the uppermost rectangle is a routine calculation. Now, diagram (a) commutes because:

$$
\begin{array}{ll}
 & \phi_1 \cdot \psi_1 \\
= & \quad \{\ \phi_1 \text{ and } \psi_1 \text{ definitions } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id}) \\
 & \cdot (\mathsf{a} \times \mathsf{id}) \cdot (o_p \times \mathsf{id} \times \mathsf{id} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \mathsf{a} \text{ natural } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
 & \cdot (o_p \times \mathsf{id} \times \mathsf{id} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \text{law (C.9) } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\mathsf{Ba} \times \mathsf{id}) \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
 & \cdot (o_p \times \mathsf{id} \times \mathsf{id} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \tau_r \text{ natural (C.5) and functors } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (o_p \times \mathsf{id} \times \mathsf{id} \times \mathsf{id}) \\
 & \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \tau_l \text{ natural (C.6) and functors } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{B}(o_p \times \mathsf{id} \times \mathsf{id}) \times \mathsf{id}) \cdot (\tau_l \times \mathsf{id}) \\
 & \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\Delta \times \mathsf{id} \times \mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr} \\
= & \quad \{\ \tau_r \text{ natural (C.5) } \} \\
 & \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \mathsf{B}((o_p \times \mathsf{id} \times \mathsf{id}) \times \mathsf{id}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id})
\end{array}
$$

$$\cdot(\triangle \times \mathsf{id} \times\mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\triangle \times \mathsf{id} \times\mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr}$$

$=$ { τ_l natural (C.6) }

$$\mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \mathsf{B}((o_p \times \mathsf{id} \times \mathsf{id}) \times \mathsf{id}) \cdot \tau_r \cdot (\mathsf{B}(\triangle \times\mathsf{id}) \times \mathsf{id})$$

$$\cdot(\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\triangle \times \mathsf{id} \times\mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr}$$

$=$ { τ_r natural (C.5) }

$$\mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \mathsf{B}(\mathsf{a} \times \mathsf{id}) \cdot \mathsf{B}((o_p \times \mathsf{id} \times \mathsf{id}) \times \mathsf{id}) \cdot \mathsf{B}(\triangle \times \mathsf{id} \times\mathsf{id})$$

$$\cdot\tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \cdot (\mathsf{a} \times \mathsf{id}) \cdot (\triangle \times \mathsf{id} \times\mathsf{id}) \cdot (\pi_2 \times \mathsf{id} \times \mathsf{id}) \cdot \mathsf{xr}$$

$=$ { *delay* and **;** definitions }

$$a_{\delta p;q}$$

Concerning the commutativity of diagram (b) reason as follows:

$$\mathsf{B}(\pi_2 \times \mathsf{id}) \cdot \phi_1$$

$=$ { ϕ_1 definition }

$$\mathsf{B}(\pi_2 \times \mathsf{id}) \cdot \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id})$$

$$\cdot(\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id})$$

$=$ { μ natural (C.16) }

$$\mu \cdot \mathsf{BB}(\pi_2 \times \mathsf{id}) \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id})$$

$$\cdot(\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id})$$

$=$ { τ_l natural (C.6) and functors }

$$\mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\pi_2 \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id})$$

$$\cdot(\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id})$$

$=$ { a and s natural }

$$\mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \mathsf{B}(\mathsf{id} \times \pi_2 \times \mathsf{id}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id})$$

$$\cdot(\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id})$$

$=$ { τ_r natural (C.5) }

$$\mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\mathsf{B}(\mathsf{id} \times \pi_2) \times \mathsf{id}) \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id})$$

$$\cdot(\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id})$$

$=$ { τ_l natural (C.6) }

$$\mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \mathsf{B}\pi_2 \times \mathsf{id}) \cdot (\mathsf{id} \times \tau_l \times \mathsf{id})$$

$$\cdot(\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id})$$

$=$ { law (C.13) }

$$\mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times \pi_2 \times \mathsf{id}) \cdot (\mathsf{id} \times (\mathsf{id} \times a_p) \times \mathsf{id})$$

$$
\begin{aligned}
= & \quad \{ \times \text{ cancellation} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{B}(\mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a}) \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times (a_p \cdot \pi_2) \times \mathsf{id}) \\
= & \quad \{ \text{functors and } \phi_2 \text{ definition} \} \\
& \phi_2 \cdot (\mathsf{id} \times \pi_2 \times \mathsf{id})
\end{aligned}
$$

And finally, to check diagram (c) compute:

$$
\begin{aligned}
& \phi_3 \cdot \mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a} \\
= & \quad \{ \phi_3 \text{ definition} \} \\
& \delta_l \cdot (a_p \times a_q) \cdot \mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a} \\
= & \quad \{ \delta_l \text{ definition} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \tau_r \cdot (a_p \times a_q) \cdot \mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a} \\
= & \quad \{ \times \text{ functor and } \tau_r \text{ natural (C.5)} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \tau_r \cdot (a_p \times \mathsf{id}) \cdot \mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a} \\
= & \quad \{ \mathsf{a} \text{ and } \mathsf{s} \text{ natural} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \tau_r \cdot \mathsf{a} \cdot \mathsf{s} \cdot \mathsf{a} \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
= & \quad \{ \text{law (C.8)} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{Ba} \cdot \tau_r \cdot (\tau_r \times \mathsf{id}) \cdot \mathsf{s} \cdot \mathsf{a} \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
= & \quad \{ \mathsf{s} \text{ natural} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{Ba} \cdot \tau_r \cdot \mathsf{s} \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
= & \quad \{ \text{law (C.7)} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{Ba} \cdot \mathsf{Bs} \cdot \tau_l \cdot (\mathsf{id} \times \tau_r) \cdot \mathsf{a} \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
= & \quad \{ \text{law (C.46)} \} \\
& \mu \cdot \mathsf{B}\tau_l \cdot \mathsf{B}(\mathsf{id} \times a_q) \cdot \mathsf{Ba} \cdot \mathsf{s} \cdot \mathsf{a} \cdot \tau_r \cdot (\tau_l \times \mathsf{id}) \cdot (\mathsf{id} \times a_p \times \mathsf{id}) \\
= & \quad \{ \text{functors and } \phi_2 \text{ definition} \} \\
& \phi_2
\end{aligned}
$$

□

APPENDIX E

A Brief Introduction to CHARITY

1. Introduction

1. CATEGORICAL PROGRAMMING. CHARITY [CF92] is an experimental implementation of a programming language entirely based on categorical data types (see section 4 in chapter 3). This means that programs are built by composition of combinators (like *anamorphisms* or *catamorphisms*) arising as universal arrows associated to such datatypes.

CHARITY makes few assumptions on the underlying category: distributivity is assumed and the Cartesian closedness requirement, implicit in the original approaches to categorical data types, namely in the [Wra88] refinement of Hagino's work [Hag87b], replaced by the assumption that all datatypes are *strong* (§3.50). CHARITY primitive types are, then, the nullary (denoted by `1`) and binary product types (denoted by the infix operator `*`, with projections `p0` and `p1`). The absence of exponentials at the core level of the language endows CHARITY with a rather different flavour when compared to more traditional functional languages. In particular, functions are not values. Moreover, function composition, instead of function application, is taken as the fundamental primitive in the language. This does not mean, however, that CHARITY lacks support for higher-order types: simply they have to be explicitly declared (see §12 in the sequel).

In this context, CHARITY may be classified as a polymorphic, strongly-typed language, which is functional in style. In particular, any program has a guarantee of 'termination', in the sense that the term representing it always reduces to a head normal form and, therefore, a 'response' is produced. Such a 'response' is computed either lazily or eagerly depending on the types involved being coinductive or inductive, respectively. In any case, the type system avoids the possibility of writing functions that may never terminate. As argued in [CF92], the semantics of CHARITY is *compatible with this mathematical intuition* [that functions are total]. In a broader sense, the language may be seen as a concept-proof for the 'slogan' in [CF92],

> [think of] *category theory as a medium of computation rather than as a computational model.*

Although both data and programs can be expressed in a pointfree way in terms of such categorical combinators, programming at such a level becomes rather awkward (namely, by the number of projections often needed to distribute variables along an expression). CHARITY programs are written in a term logic for Cartesian categories enriched with a definitional mechanism for inductive and coinductive strong data types. This allows the use of variables in combinator expressions and pattern matching. The term logic is formally defined and proved equivalent to the corresponding combinator theory in [CS95].

2. APPENDIX OVERVIEW. The following paragraphs provide a brief introduction to programming in CHARITY, emphasising the close correspondence with categorical data types as presented in chapter 3. This explains why the presentation is somewhat different from standard CHARITY presentations such as [CF92, Sch97]. On the other hand, the presentation is intended to support the use of CHARITY for prototyping some of the constructions proposed in the thesis. Actually, what makes CHARITY an interesting alternative for prototyping purposes, when compared with other declarative languages, is the very general way it provides for defining datatypes as *algebras* or *coalgebras* for functors and enforcing a discipline for their use.

2. Coinductive Types

3. DEFINITION. Let A be an arbitrary type. Infinite sequences of A, usually written as A^ω, were introduced in §3.43 as a well known example of a coinductive type. This means that A^ω represents the carrier of the final coalgebra (§3.33) for functor $\mathsf{T}_A X = A \times X$, whose dynamics consists of two observers, accessing, respectively, the $head$ and $tail$ of the sequence. In CHARITY the declaration

```
data  S -> stream A =
      head: S -> A  |  tail: S -> S.
```

introduces `stream A` as the final coalgebra

$$\langle A^\omega, \langle \mathtt{head}, \mathtt{tail} \rangle : A^\omega \longrightarrow \mathsf{T}_A\, A^\omega \rangle$$

The declaration makes the names and arities of the observers explicit, thus revealing the shape of functor T_A. Moreover, `stream` can also be thought of as the associated (strong) cotype functor $\underline{\mathsf{N}}_{\mathsf{T}_A}$ (§3.63). In general, the declaration of a coinductive type in CHARITY has the following format:

$$\begin{array}{ll}
\multicolumn{2}{l}{\texttt{data}\ S\texttt{->}T\texttt{(}A\texttt{) =}} \\
 & o_1 \texttt{:} S \texttt{->} E_1(A,S) \\
| & \ldots \\
| & o_n \texttt{:} S \texttt{->} E_n(A,S)\,\texttt{.}
\end{array}$$

This introduces coinductive type $T\texttt{(}A\texttt{)}$, parametric on A. The declaration format conveys the idea that morphisms from any type S to $T\texttt{(}A\texttt{)}$ are solely determined by morphisms from S to each $E_i(A,S)$, the output type of observer o_i. Formally, this defines $T\texttt{(}A\texttt{)}$ as the final coalgebra for a functor T_A determined by a signature of observers,

$$\langle \nu_{\mathsf{T}_A}, \langle o_1, \ldots o_n \rangle : \nu_{\mathsf{T}_A} \longrightarrow \prod_{i=1}^{n} E_i(A, \nu_{\mathsf{T}_A}) \rangle$$

Note that each o_i identifies one such observer whose type is obtained by setting $S = T\texttt{(}A\texttt{)}$ in the declaration. Therefore, $T\texttt{(}A\texttt{)}$ is $\nu_{\mathsf{T}_\mathsf{A}}$ and T itself denotes the cotype functor.

4. THE UNFOLD COMBINATOR. The basic combinator associated to a strong coinductive type is *unfold*, *i.e.*, the 'strong version' of anamorphism, denoted by unfold in §3.56. This is specified in CHARITY by supplying, for each observer o_i, the corresponding component p_i of the source coalgebra. As expected, as we are working on a strong setting, each p_i is typed as $p_i : S \times C \longrightarrow E_i(A,S)$, assuming S as the carrier of the source coalgebra and C the context type. The concrete syntax for the *unfold* expression is as follows:

$$\begin{array}{lll}
\multicolumn{3}{l}{\texttt{(}s\texttt{,}c\texttt{) =>}} \\
\texttt{(|} & s \texttt{=>}\ o_1 \texttt{:} p_1\ \texttt{(}s\texttt{,}c\texttt{)} & \\
| & \qquad\vdots & \\
| & \qquad o_n \texttt{:} p_n\ \texttt{(}s\texttt{,}c\texttt{)} & \texttt{|)}\ s
\end{array}$$

where s and c denote variables of type S and C, respectively. Let us consider some examples. In the first one, the stream of even natural numbers is generated by anamorphism $[\![\langle \mathsf{id}_{\texttt{nat}}, +_2 \rangle]\!]$ starting with 0 as seed value. Therefore, `even` turns out to be a constant of `stream nat`.

```
def even: 1  -> stream nat
    = () => (| n => head: n
             |      tail: add(n,two)
             |) zero.
```

Our second example is a function gen, which generates any stream of type X starting at a given value, in which the generator of the 'next' value is itself supplied as an additional parameter. This illustrates another feature of the language. Although 'functions' and 'values' are sharply distinguished in CHARITY, there is a mechanism for passing functions as arguments of other functions. Such function arguments are always placed between curly brackets, before the value arguments.

```
def gen{next: X -> X}: nat  -> stream X
    = m => (| n => head: n
            |      tail: next n
            |) m.
```

Therefore, we may redefine even as

```
def even: 1 -> stream nat
    = () => gen{addtwo} zero.
```

where addtwo: nat -> nat = n => add(n,two). Similarly, the following examples define the stream of all naturals and a generator of any ascending stream of naturals starting at a given number.

```
def fromzero: 1 -> stream nat
    = () => gen{succ} zero.

def from: nat -> stream nat
    = n => gen{succ} n.
```

Finally consider an example of an *unfold* with non trivial context. The function until generates an ascending stream of naturals which, from some point on, remains stable, infinitely repeating the last number. This point is, in fact, supplied as context information (lt is the 'less than' order on ℕ):

```
def until: nat * nat -> stream nat
    = (m,n) => (| x => head: {true => x
                             |false => n} (lt(x,n))
                |      tail: {true => succ x
                             |false => n} (lt(x,n))
                |) m.
```

The last example is a somewhat unusual specification of $n!$. By unfolding, the function generates a stream of factorials in which the sought value can be looked up.

```
def genfac: nat * nat -> stream nat
    = (n,m) => (| (x,y) => head: x
                |          tail: (mul(x,y), succ y)
                |) (n,m).

def factorial: 1 -> stream nat
    = () => genfac (one,one).
```

The corresponding diagram is

$$\begin{tikzcd}
\mathtt{stream\ nat} \arrow[rr, "\langle \mathtt{head},\mathtt{tail}\rangle"] & & \mathtt{nat}\times\mathtt{stream\ nat} \\
\mathtt{nat}\times\mathtt{nat} \arrow[u, "\mathtt{genfac}{=}[\!(g)\!]"] \arrow[rr, "g{=}\langle\pi_1{,}\langle\mathtt{mul}{,}\mathtt{succ}\cdot\pi_2\rangle\rangle"'] & & \mathtt{nat}\times(\mathtt{nat}\times\mathtt{nat}) \arrow[u, "\mathsf{id}\times[\!(g)\!]"'] \\
\mathtt{1} \arrow[u, "\langle\underline{\mathsf{one}}{,}\underline{\mathsf{one}}\rangle"]
\end{tikzcd}$$

5. THE RECORD COMBINATOR. An element $\underline{e}$ of the final coalgebra, *i.e.*, a *point* (§2.7) $\underline{e} : \mathbf{1} \longrightarrow \nu_{\mathsf{T}_\mathsf{A}}$ can be specified by its behaviour at each observer, *i.e.*, by prescribing values for all possible observations. For example, a stream of naturals starting with zero and followed by all the even numbers can be defined as the unique arrow zeven making the following diagram to commute:

$$\begin{tikzcd}
\mathtt{stream\ nat} \arrow[r, "\langle \mathtt{head},\mathtt{tail}\rangle"] & \mathtt{nat}\times\mathtt{stream\ nat} \\
\mathbf{1} \arrow[u, "\mathtt{zeven}"] \arrow[ur, "\langle\underline{\mathsf{one}}{,}\mathsf{even}\rangle"']
\end{tikzcd}$$

Therefore,

$$\begin{array}{ll}
 & \langle \mathtt{head},\mathtt{tail}\rangle\cdot\mathtt{zeven} \;=\; \langle \underline{\mathtt{one}},\mathtt{even}\rangle \\
\equiv & \qquad \{\times \text{ fusion}\} \\
 & \langle \mathtt{head}\cdot\mathtt{zeven},\mathtt{tail}\cdot\mathtt{zeven}\rangle \;=\; \langle \underline{\mathtt{one}},\mathtt{even}\rangle \\
\equiv & \qquad \{\text{ equality}\} \\
 & \mathtt{head}\cdot\mathtt{zeven} \;=\; \underline{\mathtt{one}} \wedge \mathtt{tail}\cdot\mathtt{zeven} \;=\; \mathtt{even}
\end{array}$$

CHARITY provides a special syntax for describing this construction and denotes by *record* the corresponding combinator. The zeven sequence is defined in this concrete syntax as

```
def zeven: 1 -> stream nat
    = () => (head: one, tail: even).
```

which reads exactly as ‘the head of `zeven` is `one` and its tail is `even`’. In presence of context, **1**, in the diagram above, is replaced by $\mathbf{1} \times C \cong C$, for C a context type, and the values for the observers may depend on C. What is specified, in this case, is a *generalized* element (§2.7) of `stream nat`. As an example, consider the following function which generates an infinite sequence of naturals starting at a given number n, with n repeated in the first two positions:

```
def nfrom: nat -> stream nat
    = n => (head: n, tail: from n).
```

The same construction can be applied to any coinductive type. Thus, a way of populating such types is provided by specifying particular values for the observers. The general pattern for the *record* combinator is thus

$$\begin{array}{l} c \Rightarrow \\ \quad (\ \ o_1 : f_1\,(c) \\ \quad \mid\ \ \vdots \\ \quad \mid\ \ o_n : f_n\,(c)\ \ ) \end{array}$$

where each f_i is a function from the context type to the codomain of the observer o_i. This is the same device which the CHARITY runtime system uses to print out coinductive datatypes, providing a rather verbose, but clear and uniform, presentation. We may recall, at this point, that coinductive types are evaluated lazily. Therefore, the interpreter only displays that part of the structure which has already been demanded. User interaction is required to simulate demand.

6. THE MAP COMBINATOR. For each coinductive type, the *map* combinator denotes the action on morphisms, with strength, of the corresponding cotype functor, as explained in §3.62. As an example, consider the following two functions on streams. The first one increments a stream of natural numbers, illustrating a *map* with context. The second transforms a stream of A into a stream of A-pairs by replicating each element.

```
def incr: stream nat * nat  -> stream nat
    = (s,n) => stream{ x => add(x,n)} s.

def diagst: stream A  -> stream(A * A)
    = s => stream{ x => (x,x)} s.
```

Note the *map* combinator is specified by the name of the cotype functor, `stream` in these examples, and the function to be applied enclosed between curly brackets as it is always the case with 'function' arguments in CHARITY (recall §4).

Streams are just parametrized by one type A. Therefore, the argument to the *map* expression over streams is just a function from A to another type B or, in the presence of a context type C, from $A \times C$ to B. However, coinductive (as well as inductive) types can be specified in CHARITY with several different parameters. For example, consider the following type, parametric on A and B.

```
data S -> bistream(A,B) =  dt: S -> A  |  ct: S -> B
                        |  lf: S -> S  |  rg: S -> S.
```

Note `bistream(A,B)` has two pure observers, observing on A and B, and two transformers providing a binary branching structure. The associated coinductive type can easily be recognised as the class of infinite binary trees carrying two kinds of information on their nodes. For illustration purposes, one may think of data and control information. Consider, for example, a function `flow` generating an infinite tree from which we may observe a natural number (thought of as the 'data' part) and a Boolean (representing, say, control information):

```
def flow: (nat * bool) * nat -> bistream(nat, bool)
    = ((init, control), critical) =>
          (| (n,b) => dt: n
           |          ct: and(b, lt(n, critical))
           |          lf: (add (n, one), b)
           |          rg: (add (n, two), not b)
           |) (init, control).
```

Pursuing observation always on the left (resp. right), we retrieve the odd (resp. even) natural numbers and in alternative branches the flag used to compute the control information appears complemented. Finally, notice how the 'control' value is computed using some 'extra' information (*e.g.*, a 'critical' reference level) provided as context to *unfold*.

To *map* over a bistream, requires the specification of a transformation for each parameter, *i.e.*,

$$\begin{aligned} h_A : A \longrightarrow A' \qquad &\text{or} \quad h_A : A \times C \longrightarrow A' \\ h_B : B \longrightarrow B' \qquad &\text{or} \quad h_B : B \times C \longrightarrow B' \end{aligned}$$

For example, a bistream generated by `flow` can be uniformly modified by incrementing the data values and trivialising the control information. The two effects can be achieved simultaneously by the following function:

```
def modify_flow: bistream(nat, bool) -> bistream(nat, 1)
    = t => bistream{n => succ n, b => ()} t.
```

Technically, `bistream` is simply the cotype functor associated to $\mathsf{R}_{\mathsf{A}\times\mathsf{B}}X = A \times B \times X^2$. The general format of the *map* combinator over a type $T\,(A_1,\ \ldots,A_m)$ with m parameters, is

$$\begin{array}{l} (t,c)\ \texttt{=>} \\ \quad T\ \{\ x_1\ \texttt{=>}\ \ h_1\,(x_1,c) \\ \quad\ \ \mid\ \vdots \qquad\qquad\quad \vdots \\ \quad\ \ \mid\ x_m\ \texttt{=>}\ h_m\,(x_m,c)\ \ \}\ t \end{array}$$

where $h_i : A_i \times C \longrightarrow A_i'$, yielding a result of type $T\,(A_1',\ \ldots,A_m')$.

3. Inductive Types

7. DEFINITION. Inductive types are declared in CHARITY as

$$\begin{array}{l} \texttt{data}\ T\,(A)\ \texttt{->}\ S\ \texttt{=} \\ \qquad c_1 : E_1(A,S)\ \texttt{->}\ S \\ \quad\ \mid\ \ldots \\ \quad\ \mid\ c_n : E_n(A,S)\ \texttt{->}\ S\,\texttt{.} \end{array}$$

which introduces type $T\,(A)$, parametric on A, as the initial algebra (§3.14)

$$\langle \mu_{\mathsf{T}_A}, [c_1,\ldots,c_n] : \sum_{i=1}^{n} E_i(A,\mu_{\mathsf{T}_A}) \longrightarrow \mu_{\mathsf{T}_A}\rangle$$

Moreover, T acts as the associated type functor $\underline{\mathsf{M}}_{\mathsf{T}_A}$ (§3.63). Mapping a function over an inductive type corresponds exactly to the application of the action on morphisms of its type functor. As expected, in the CHARITY term logic the syntax of the *map* expression (§6) for both inductive or coinductive types coincide.

The natural numbers, Boolean values and sequences of A are typical examples of inductive types included in the CHARITY standard prelude. Another equally relevant case is the datatype of 'partial elements' of A also called the 'maybe' or 'success-failure' datatype, which defined as the coproduct of A with the final type **1**. Declarations of these types in CHARITY follow:

```
data  nat -> S      =  zero: 1 -> S | succ: S -> S.

data  bool -> S     =  true: 1 -> S | false: 1 -> S.

data  list A -> S  =  nil : 1 -> S | cons: A * S -> S.

data  SF A -> S     = ss: A -> S | ff: 1 -> S.
```

The language provides some notation (and a handful of primitive functions) for dealing with these datatypes. For example, square brackets can be used for list enumeration, as it is usual in functional languages. Finally note that the binary coproduct $A + B$ is introduced in the language as the (non recursive) inductive type

```
data coprod(A, B) -> S  =  b0: A -> S | b1: B -> S.
```

In retrospect, the *coproduct* type is a prototypical example of an inductive type. Although non recursive, it is completely determined by two constructors — the injections ι_1 and ι_2 (`b0` and `b1`, respectively). Dually, the *product* type is determined by two observers — the projections π_1 and π_2 (written `p0` and `p1`) — being, therefore, a prime example of a coinductive type.

8. THE FOLD COMBINATOR. The *fold* combinator, or 'strong catamorphism' (see §3.61 where it is denoted by fold), is specified by introducing, for each constructor c_i, the corresponding constructor, d_i, of the target algebra. Each of them is typed as $d_i : E_i(A, S) \times C \longrightarrow S$, where C is the type of the context and S the carrier of the target algebra. The target algebra is, of course, just the *either* of all such d_i. The concrete syntax for the *fold* expression is:

$$\begin{array}{lll}
\texttt{(}s\texttt{,}c\texttt{) =>} & & \\
\quad \texttt{\{|} & c_1 \texttt{:} s_1 \texttt{=>}\ d_1\ \texttt{(}s_1\texttt{,}\ c\texttt{)} & \\
\quad \texttt{|} & \vdots \qquad\qquad\quad \vdots & \\
\quad \texttt{|} & c_n \texttt{:} s_n \texttt{=>} d_n\ \texttt{(}s_n\texttt{,}\ c\texttt{)} & \texttt{|\}}\ s
\end{array}$$

where s and c denote variables of types S and C, respectively.

As an example, consider the following diagram which specifies the reduction of a list by a monoid $\langle M; \theta, u\rangle$ as a list catamorphism.

$$\begin{tikzcd}
1 + M \times \mathtt{list}M \arrow[r, "\texttt{[nil,cons]}"] \arrow[d, "\mathsf{id}_1 + \mathsf{id}_M \times (\!|\mathtt{r}|\!)"'] & \mathtt{list}M \arrow[d, "\mathtt{reduce} = (\!|\mathtt{r}|\!)"] \\
1 + M \times M \arrow[r, "\mathtt{r} = {[u, \theta]}"'] & M
\end{tikzcd}$$

The `reduce` operation is defined in CHARITY as follows:

```
def reduce{u: 1 -> M, theta: M * M -> M}: list(M) -> M
    =  l  =>  {| nil: ()     => u
               | cons: (m,n) => theta(m,n)
               |}  l.
```

Note, in particular, how the definition is parametrized by a reduction monoid.

9. THE CASE COMBINATOR. Recall from §5 that the *record* combinator provides a canonical way of specifying (generalized) elements of a coinductive type. Dually, (generalized) elements of any type S, having an inductive type as domain of variation, can arise in a simple (non recursive) way by defining its value on each constructor of the domain. For example, the 'second successor' function on the naturals is the unique arrow making the following diagram to commute:

$$\begin{tikzcd}
1 + \mathtt{nat} \arrow[r, "{[\text{zero,succ}]}"] \arrow[d, "{[\text{two,succ}\bullet\text{succ}]}"'] & \mathtt{nat} \arrow[dl, "\text{ssucc}"] \\
\mathtt{nat} &
\end{tikzcd}$$

This is written in CHARITY as

```
def ssucc: nat -> nat
    = n => { zero   => two
           | succ x => succ succ succ x
           } n.
```

or, simply, as

```
def ssucc: nat -> nat
    = zero   => two
      succ n => succ succ succ n.
```

This construction is known in CHARITY as the *case* combinator whose syntax, in the general case, is

$$
\begin{array}{ll}
\texttt{(}s\texttt{,}\,c\texttt{) =>} & \\
& \texttt{\{}\ \ c_1 s_1 \texttt{=>}\ d_1\,\texttt{(}s_1\texttt{,}\,c\texttt{)} \\
& \texttt{|}\ \vdots \qquad\qquad\qquad\qquad \vdots \\
& \texttt{|}\ \ c_n s_n \texttt{=>} d_n\,\texttt{(}s_n\texttt{,}\,c\texttt{)}\ \ \texttt{\}}\ s
\end{array}
$$

where each d_i is a function from $E_i(A, \mu_{\mathsf{T}_A}) \times C$, to the target type. Notice that $E_i(A, \mu_{\mathsf{T}_A})$ is the domain of constructor c_i and C types context information. Therefore, the domain of the *case* combinator is $\mathsf{T}_A \times C$. Using strength, context is pushed inside the T_A outermost (coproduct) structure and, therefore, even in this general case, the combinator is still determined by an *either* $[d_1, \ldots, d_n]$.

As a last example consider function `inseg` which returns a sequence corresponding to the initial segment of a natural number. The function is basically a combination of a *fold* over lists and a *case* on the naturals.

```
def inseg: nat  -> list nat
    = n => {| zero: () => []
            | succ: l  => { ff => l
                          | ss n => cons(add(one,n), l)
                          } hd l
            |} n.
```

4. Further Recursion Patterns

10. PARAMORPHISMS. *Paramorphisms* [Mee92] generalise catamorphisms in order to deal with cases in which the result of a recursive function depends not only on computations in substructures of its argument, but also on the substructures themselves. The recursion pattern it entails, in particular when defined over the natural numbers, is known as *primitive recursion*.

In the general case, a *paramorphism* is defined as the unique arrow making the diagram below to commute. The domain of the 'target algebra' is now the T image of its carrier 'packed' with the inductive type itself.

$$\begin{tikzcd}
X & \mathsf{T}\,(X \times \mu_{\mathsf{T}}) \arrow[l, "f"'] \\
\mu_{\mathsf{T}} \arrow[u, "\mathsf{par}_{\mathsf{T}}\, f"] & \mathsf{T}\,\mu_{\mathsf{T}} \arrow[l, "\alpha_{\mathsf{T}}"] \arrow[u, "\mathsf{T}\,\langle \mathsf{par}_{\mathsf{T}}\, f, \mathsf{id}\rangle"']
\end{tikzcd}$$

The diagram entails the following universal property

$$h = \mathsf{par}_{\mathsf{T}}\, f \iff h \cdot \alpha_{\mathsf{T}} = f \cdot \mathsf{T}\,\langle h, \mathsf{id}\rangle$$

The usual factorial function arises by a suitable instantiation of the diagram, taking $\mathsf{T} = \mathbf{1} + \mathsf{Id}$ and $\langle \mathbb{N}, [\underline{0}, \mathrm{succ}] \rangle$ as its initial algebra.

$$\begin{tikzcd}
\mathbb{N} & \mathbf{1} + (\mathbb{N} \times \mathbb{N}) \arrow[l, "f"'] \\
\mathbb{N} \arrow[u, "\mathsf{par}_\mathsf{T}\, f"] & \mathbf{1} + \mathbb{N} \arrow[l, "{[\underline{0},\mathrm{succ}]}"] \arrow[u, "\mathsf{id}+\langle \mathsf{par}_\mathsf{T}\, f{,}\mathsf{id} \rangle"']
\end{tikzcd}$$

for $f = [\underline{1}, \mathrm{mul} \cdot (\mathsf{id}_\mathbb{N} \times \mathrm{succ})]$.

Paramorphisms, which have no direct implementation in CHARITY, can be defined in terms of catamorphisms. Actually, a paramorphism can be defined as a composition of a projection with a cata:

$$\mathsf{par}_\mathsf{T}\, f \;=\; \pi_1 \cdot (\!|\langle f, \alpha_\mathsf{T} \cdot \mathsf{T}\; \pi_2 \rangle|\!)_\mathsf{T}$$

Thus, the 'paramorphic' version of the factorial function will be written as

```
def factorial : nat -> nat
 =  n  => p0 f_cata n.

def f_cata : nat -> nat * nat
 = n  => {| zero: () => (one, zero)
          | succ: (x,y)  => (mul(x, succ y), succ y)
          |} n.
```

The language, however, provides an easier way of dealing with functions defined as paramorphisms, by using #. Inside a *fold*, # stands for the value being currently analysed, *before* the recursive application. Our 'paramorphic' factorial will become, then,

```
def factorial : nat -> nat
    = n  => {| zero: () => one
             | succ: x  => mul (x ,succ #)
             |} n.
```

11. APOMORPHISMS. An *apomorphism* [VU97] is the formal dual to the paramorphism combinator discussed in the previous paragraph. It allows for the final result to be either generated in successive steps or 'all at once' without recursion. Therefore, the codomain of the source 'coalgebra' becomes the sum of its carrier with the coinductive type itself. The diagram is

$$\begin{tikzcd}
\nu_{\mathsf{T}} \arrow[r, "\omega_{\mathsf{T}}"] & \mathsf{T}\,\nu_{\mathsf{T}} \\
X \arrow[u, "\mathsf{apo}_{\mathsf{T}}\,p"] \arrow[r, "p"'] & \mathsf{T}\,(X + \nu_{\mathsf{T}}) \arrow[u, "{\mathsf{T}\,[\mathsf{apo}_{\mathsf{T}}\,p,\mathsf{id}]}"']
\end{tikzcd}$$

entailing the following universal property

$$h = \mathsf{apo}_{\mathsf{T}}\,p \iff \omega_{\mathsf{T}} \cdot h = \mathsf{T}\,[h, \mathsf{id}] \cdot p$$

As expected, by dualising paramorphisms, this can be reduced to an anamorphism composed with an injection, *i.e.*,

$$\mathsf{apo}_{\mathsf{T}}\,p \;=\; [\!([p, \mathsf{T}\,\iota_2 \cdot \omega_{\mathsf{T}}])\!]_{\mathsf{T}} \cdot \iota_1$$

Again, there is a way of programming with *apomorphisms* in CHARITY, even though the combinator is not directly available. The annotation `@` is used, inside an *unfold*, to write the value being generated, thus stopping recursion. As an example, consider the following function which, given a stream s of natural numbers, generates a new stream by doubling each element. However, the function sticks to this behaviour only until (and if) the value `ten` appears in the input. When this happens, `zero` is displayed and the output stream becomes just s.

```
def double_reset_at_ten : stream nat -> stream nat
    = l =>
        (| (h,t) =>
            head: { false => double h
                  | true  => zero } eq(ten, h)
            tail: { false => (head t, tail t)
                  | true  => @(head: h, tail: t)} eq(ten, h)
         |) (head l, tail l).
```

5. Higher-Order Types

12. DEFINITION. Although CHARITY requires only the working category to be distributive (§1), higher-order types may be introduced via a generalisation, on the coinductive side, of the declaration mechanism [Sch97]. This becomes

$$\begin{array}{ll} \texttt{data}\ S\,\texttt{->}\,T\texttt{(}A\texttt{)}\ \texttt{=} & \\ & o_1 : S\,\texttt{->}\,F_1(A)\,\texttt{=>}\,E_1(A,S) \\ \mid & \ldots \\ \mid & o_n : S\,\texttt{->}\,F_n(A)\,\texttt{=>}\,E_n(A,S)\,\texttt{.} \end{array}$$

allowing each observer o_i to be typed as a function from $F_i(A) \times S$ to $E_i(A, S)$. Therefore, the type which is implemented by this declaration is

$$\langle \nu_{\mathsf{T}_\mathsf{A}}, \langle o_1, \ldots o_n \rangle : \nu_{\mathsf{T}_A} \longrightarrow \prod_i E_i(A, \nu_{\mathsf{T}_A})^{F_i(A)} \rangle$$

Note that recursion parameters always appear in covariant positions. The associated combinators remain unaltered, but for the case of *map*, whose ‘function’ argument has to be applied both co- and contra-variantly.

A typical example of a higher-order type introduced in this way is the exponential type itself:

```
data  S -> exp(A,B) =  fn: S -> A => B.
```

The type is parametric on A and B and corresponds to B^A. Note that observer `fn` is just the ev function (§2.33). Therefore, CHARITY will compute `six:nat` when evaluating

```
fn (three, (fn: n => factorial n)).
```

Also notice that

```
(fn: n => factorial n)
```

is a *record* (§5) expression, which, in this case, encapsulates a function f : `A` $\longrightarrow$ `B` as a value of type `exp(A,B)`. Finally, function composition is defined as

```
def compose: exp(A, B) * exp(B, C) -> exp(A, C)
    = ((fn: f), (fn: g))  => (fn: a => g f a).
```

13. A DETAILED EXAMPLE. Being non recursive, *unfold* degenerates into the *record* combinator for the exponential type. This is, of course, not the case of more elaborated higher-order types, such as the ones used in this thesis for prototyping components. As those are covered in the main text, this introduction to CHARITY ends discussing a possible coinductive encoding of *sets* and *partial functions* as non recursive coinductive types (see [Bar99] for details).

14. SETS AND MAPPINGS. Inductive, tree-like structures, are widely used in functional programming. In the 'formal methods' community, however, the data modelling primitive selected as 'first choice', when facing a design problem, is most likely to be some kind of *mapping*, in order to express functional dependences which pervade most information system models. This entails a subtle, but expressive, shift of perspective. Notice a *set* can also be thought of as a special (degenerate) case of a *mapping*, or *partial function*, via the isomorphism $\mathcal{P}A \cong A \rightharpoonup \mathbf{1}$.

'Unordered' structures, such as maps or sets are easier to observe than to construct (in an effective computational sense). Of course, they have a more or less obvious implementation in functional languages as certain kinds of sequences. But we will look here for more direct representations.

15. REPRESENTING SETS. To begin with, consider the following representation of sets by their characteristic functions:

```
data S -> set A = in: S -> A => bool.
```

From this perspective, a set is regarded as a structure accessed, or observed, by a predicate (the observer `in`) encoding set membership: all one is able to know about a set is whether a particular value belongs to it. Sets, even if not finite, can be easily defined in this way. For example,

```
def natset: 1 -> set nat
    =  ()  =>  (in: x => true).

def evens: 1  -> set nat
    =  ()  =>  (in:  x => and(member(x,natset), even x) ).
```

Simple operations on sets are defined as operations on predicates, as in, for example, the following representation of the *empty set*, *union*, *intersection*, *difference* and *ZF comprehension*.

```
def empty : 1 -> set A
    = () => (in: x => false).

def union: set A * set A -> set A
    = (s1,s2) => (in: x => or(in(x,s1),in(x,s2))).

def diff: set A * set A -> set A
    = (s1,s2) => (in: x => and(in(x,s1), not in(x,s2))).

def zf{pred: A -> bool}: set A -> set A
    = s  => (in: x => and (in(x,s), pred x) ).
```

As datatype `set A` is parametric in `A`, some operations require a specific definition of equality on `A`, as is the case of *singleton set* formation:

```
def sing{equal: A * A -> bool}: A -> set A
    = a => (in: x => equal(a,x)).
```

Finally, for any function `f` from `A` to `B`, `set{f}` will denotes $\mathcal{P}f$.

There are, however, some familiar operations over sets that can not be programmed in this way, as they rely on (the axiom of) choice. Computationally, this means that they presume the existence of an ordered representation of an universe with respect to which the sets of interest are defined. This is typically the case of set equality, subset inclusion and monoidal reductions: all of them require the ability to pick an element from a set. We shall thus isolate the choice dependent operations and provide a separate specification of a universe — `U(A)` —- and a choice function over it. Sequences offer a simple implementation of `U(A)`, but in CHARITY one is not limited to finite structures. Therefore, a possible encoding of choice is made over possible infinite lists $A^\infty = A^\star \cup A^\omega$. Such partial streams are called *colists* of A in [CF92], where the prefix *co* identifies the *possibly infinite version* of an inductive type obtained simply by arrow reversing. Colists are declared as

```
data S -> colist(A) = delist: S -> SF(A * S).
```

Taking `U(A)` as `colist(A)`, it is useful to have some generation functions for possible 'universes'. Two possibilities are:

```
def col_gen{f: A -> A}: A -> colist(A)
    = x => (| a => delist: ss (a, f a)  |) x.

def col_genwhile{f: A -> A, cond: A -> bool}: A -> colist(A)
    = x => (| a => delist: { true  => ss (a, f a)
                           | false => ff
                           } cond a
            |) x.
```

Finally, a choice function, accepting an additional filter predicate, is provided,

```
def choice{pred}: set A * U A -> SF A
  = (s, u) => col_first_st{x => and(in(x,s), pred(x))} u.
```

where function `col_first_st` returns the first element of a colist satisfying a given predicate.

A paradigmatic example of a function depending on choice is the `exists` predicate

```
def exists{pred: A -> bool}: set A * U A -> bool
    = (s,u)  => { ff    => false
                | ss a  => true
                } choice{pred}(s,u).
```

which is used in the definition of, *e.g.*, subset inclusion and test for disjointness:

```
def subseteq: set A * set A * U A -> bool
    = ((s1,s2),u)  =>
         not exists{ x => not member(x,s2) } (s1,u).

def eqsets: (set A * set A) * U A -> bool
    = ((s1,s2),u)  =>
         and(subseteq((s1,s2),u), subseteq((s2,s1),u)).

def disjoint: (set A * set A) * U A -> bool
    = ((s1,s2),u)  =>  eqsets ((inter(s1,s2), empty), u).
```

Note the code for all those functions on `set A` which require choice, is actually parametric wrt to the 'universe' specification. This lower level is just supposed to deliver the function *choice*.

16. REPRESENTING MAPPINGS. A direct implementation of *mappings*, or *partial functions*, also as a coinductive datatype, follows the same strategy. The space of mappings from A to B, written as $A \rightharpoonup B$, is defined by

```
data S -> maps(A, B) = ap: S -> A => SF B.
```

stating that elements of $A \rightharpoonup B$ are observed through evaluation (the observer `ap`, for "apply"), which may return an undefined value. The identity mapping is defined by a *case* expression and composition amounts to the Kleisli composition for the 'maybe' monad (§A.7).

```
def mid: set A  -> maps(A, A)
    = s => (ap: a => { true => ss a | false => ff } in(a,s)).

def mcomp: maps(A, B) * maps(B, C) -> maps(A, C)
    = ((ap: t), (ap: r))  => (ap: a => kleisli{t,r} a).
```

where the definiton of `kleisli` is given in §A.13.

Typical operations over maps, such as the ones in the meta-language of VDM, are easily supported. For example, consider the following definitions of *overwrite* and *domain restriction*.

```
def over: maps(A,B) * maps(A,B) -> maps(A,B)
    = (m1,m2) => (ap: x => { ff     => ap(x,m1)
                           | _      => ap(x,m2)
                           } ap(x,m2)  ).


def dr: maps(A,B) * set A  -> maps(A,B)
    = (m,s) =>  (ap: x => { true  => ap(x,m)
                          | false => ff
                          } in(x,s)  ).
```

Also note that, being parametric in two arguments, its action on morphisms is divided in three cases: acting on the domain through `f`, `maps{f,x => x}`, on the range through `g`, `maps{x => x,g}`, or both, `maps{f,g}`.

In traditional formal specification methods one is used to restrict oneself to *finite* maps. Such restriction, however, is not essential for this CHARITY implementation. The following is an example of an infinite map which maps every even natural number to its successor.

```
def evsucc: 1 -> maps(nat,nat)
    = () => (ap: n =>
        {true => ss succ n  | false => ff} even n ).
```

Bibliography

[ABNO97] J. J. Almeida, L. S. Barbosa, F. L. Neves, and J. N. Oliveira. CAMILA: Prototyping and refinement of constructive specifications. In M. Johnson, editor, *6th Int. Conf. Algebraic Methods and Software Technology (AMAST)*, pages 554–559, Sydney, December 1997. Springer Lect. Notes Comp. Sci. (1349).

[Abr94] S. Abramsky. Interaction categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 1–15. Prentice-Hall International, 1994.

[Abr96] J. R. Abrial. *The* B *Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[AC96] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[Ace92] L. Aceto. *Action Refinement in Process Algebras*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1992. (PhD thesis, University of Sussex (1990)).

[Acz88] P. Aczel. *Non-Well-Founded Sets*. CSLI Lecture Notes (14), Stanford, 1988.

[Acz93] P. Aczel. Final universes of processes. In Brooks et al, editor, *Proc. Math. Foundations of Programming Semantics*. Springer Lect. Notes Comp. Sci. (802), 1993.

[Acz97] P. Aczel. Lectures on semantics : The initial algebra and final coalgebra perspectives. In H. Schwichtenberg, editor, *Logic of Computation*. Springer-Verlag, 1997. Lectures for the 1995 Marktoberdorf School on Logic of Computation.

[Ada00] J. Adamek. Final colagebras as ideal completions of initial algebras. Talk at the MFIT summer school on algebraic and coalgebraic methods in mathematics of program construction, Lincoln College, Oxford University, April 2000.

[AG97] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM TOSEM*, 6(3):213–249, 1997.

[AGN94] S. Abramsky, S. Gay, and R. Nagarajan. Interaction categories and the foundation of typed concurrent programming. In M. Broy, editor, *Deductive Program Design: Proc. of the 1994 Marktoberdorf Summer School*. NATO ASI Series F, Springer Verlag, 1994.

[AJ94] S. Abramsky and S. Jagadeeson. New foundations for the geometry of interaction. *Information & Computation*, 111:53–119, 1994.

[AM88] P. Aczel and N. Mendler. A final coalgebra theorem. In D. Pitt, D. Rydeheard, P. Dybjer, A. Pitts, and A. Poigne, editors, *Proc. Category Theory and Computer Science*, pages 357–365. Springer Lect. Notes Comp. Sci. (389), 1988.

[Aug93] A. Augusteijn. *Functional programming, program transformations and compiler construction*. PhD thesis, Department of Computing Science, Eindhoven University of Technology, The Netherlands, 1993.

[AV95] S. Abramsky and S. Vickers. Quantales, observation logic and process semantics. *Math. Struct. in Comp. Sci.*, 3:161–227, 1995.

[Bac78] J. Backus. Can programming be liberated from the Von Neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21:613–641, 1978.

[Bac88] R. Backhouse. An exploration of the Bird-Meertens formalism. CS 8810, Groningen University, 1988.

[Bae97] J. C. Baez. An introduction to n-categories. In E. Moggi and G. Rosolini, editors, *Proc. 7th Conf. Category Theory and Computer Science*. Springer Lect. Notes Comp. Sci. (1290), 1997.

[Bal00] M. Baldamus. Compositional constructor interpretation over coalgebraic models for the π-calculus. In H. Reichel, editor, *CMCS'00 - Workshop on Coalgebraic Methods in Computer Science*. ENTCS, volume 33, Elsevier, 2000.

[Bar70] M. Barr. Coequalizers and cofree cotriples. *Mathematische Zeitschrift*, 166:307–322, 1970.

[Bar92] L. S. Barbosa. Sobre a especificação matemática de sistemas concorrentes. PAPCC Thesis DI-LSB-92:9:1, DI (U. Minho), September 1992. (in portuguese).

[Bar93] M. Barr. Terminal coalgebras in well-founded set theory. *Theor. Comp. Sci.*, 114(2):299–315, 1993.

[Bar99] L. S. Barbosa. Prototyping processes. In M. C. Meo and M. Vilares Ferro, editors, *Proc. of AGP'99 - Joint Conference on Declarative Programming*, pages 513–527, L'Aquila, Italy, 6-9 September 1999.

[Bar00] L. S. Barbosa. Components as processes: An exercise in coalgebraic modeling. In S. F. Smith and C. L. Talcott, editors, *FMOODS'2000 - Formal Methods for Open Object-Oriented Distributed Systems*, pages 397–417. Kluwer Academic Publishers, September 2000.

[Bar01a] L. S. Barbosa. Process calculi *à la* Bird-Meertens. In *CMCS'01 - Workshop on Coalgebraic Methods in Computer Science*, pages 47–66, Genova, April 2001. ENTCS, volume 44.4, Elsevier.

[Bar01b] F. Bartels. Generalised coinduction. In *CMCS'01 - Workshop on Coalgebraic Methods in Computer Science*, pages 67–87, Genova, April 2001. ENTCS, volume 44.4, Elsevier.

[BB87] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language Lotos. *Computer Networks and ISDN Systems*, 14, 1987.

[BBB$^+$85] F. L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, and H. Wössner. *The Munich Project* CIP. *Volume I: The Wide Spectrum Language* CIP-L. Springer Lect. Notes Comp. Sci. (183), 1985.

[BCS98] R. Blute, J. Cockett, and R. Seely. Feedback for linearly distributive categories: Traces and fixpoints. Presented on the celebration of W. Lawvere 60th birthday (submitted for publication, available from `triples.math.mcgill.ca/~rags/linear/`), 1998.

[BD99] H. Bowman and J. Derrick. A junction between state based and behavioural specification. In P. Ciancarini, A. Fantechi, and R. Gorrieri, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 213–239. Kluwer Academic Publishers, February 1999.

[Ben67] J. Benabou. Introduction to bicategories. *Springer Lect. Notes Maths. (47)*, pages 1–77, 1967.

[BG92] S. Brookes and S. Geva. Computational comonads and intensional semantics. In M. Fourman, P. Johnstone, and A. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 1992.

[BH93] R. C. Backhouse and P. F. Hoogendijk. Elements of a relational theory of datatypes. In B. Möller, H. Partsch, and S. Schuman, editors, *Formal Program Development*, pages 7–42. Springer Lect. Notes Comp. Sci. (755), 1993.

[Bir35] G. Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

[Bir87] R. S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, volume 36 of *NATO ASI Series F*, pages 3–42. Springer-Verlag, 1987.

[Bir98] R. Bird. *Functional Programming Using Haskell*. Series in Computer Science. Prentice-Hall International, 1998.

[BJJM98] R. C. Backhouse, P. Jansson, J. Jeuring, and L. Meertens. Generic programming: An introduction. In S. D. Swierstra, P. R. Henriques, and J. N. Oliveira, editors, *Third International Summer School on Advanced Functional Programming, Braga*, pages 28–115. Springer Lect. Notes Comp. Sci. (1608), September 1998.

[BM87] R. S. Bird and L. Meertens. Two exercises found in a book on algorithmics. In L. Meertens, editor, *Program Specification and Transformation*, pages 451–458. North-Holland, 1987.

[BM94] R. S. Bird and O. de Moor. Relational program derivation and context-free language recognition. In A. W. Roscoe, editor, *A Classical Mind: Essays dedicated to C.A.R. Hoare*, pages 17–35. Prentice Hall International, 1994.

[BM96] J. Bairwise and P. Moss. *Vicious Circles*. CSLI Lecture Notes (59), Stanford, 1996.

[BM97] R. Bird and O. Moor. *The Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997.

[Bor94a] F. Borceux. *Handbook of Categorial Algebra (3 volumes)*. Cambridge University Press, 1994.

[Bor94b] F. Borceux. *Handbook of Categorial Algebra (vol. 2)*. Cambridge University Press, 1994.

[BW85] M. Barr and C. Wells. *Toposes, Triples and Theories*. Springer-Verlag, 1985.

[BW90a] M. Barr and C. Wells. *Category Theory for Computer Scientists*. Series in Computer Science. Prentice-Hall International, 1990.

[BW90b] J. Beaten and W. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[Car87] A. Carboni. Bicategories of partial maps. *Cahiers Top. - Géom. Diff. - Cat.*, 28(2):111–126, 1987.

[CF92] R. Cockett and T. Fukushima. About Charity. Yellow Series Report No. 92/480/18, Dep. Computer Science, University of Calgary, June 1992.

[Cir98] C. Cirstea. Coalgebra semantics for hidden algebra. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques*, pages 174–189. Springer Lect. Notes Comp. Sci. (1376), 1998.

[CLW93] A. Carboni, S. Lack, and R. F. C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84:145–158, 1993.

[Coc93] R. Cockett. Introduction to distributive categories. *Mathematical Structures in Computer Science*, 3(3):277–307, 1993.

[CoF95] CoFI. The CoFI algebraic specification language, tentative design: Language summary. BRICS ns-96-15, BRICS, Aarhus University, 1995.

[CPW98] G. L. Cattani, A. J. Power, and G. Winskel. A categorical axiomatics for bisimulation. In *Proc. CONCUR' 98*, pages 581–596. Springer Lect. Notes Comp. Sci. (1466), 1998.

[CR97] G. Costa and G. Reggio. Specification of abstract dynamic data types: A temporal logic approach. *Theor. Comp. Sci.*, 173(2), 1997.

[CS92] R. Cockett and D. Spencer. Strong categorical datatypes I. In R. A. G. Seely, editor, *Proceedings of Int. Summer Category Theory Meeting, Montréal, Québec, 23–30 June 1991*, pages 141–169. AMS, CMS Conf. Proceedings 13, 1992.

[CS95] R. Cockett and D. Spencer. Strong categorical datatypes II: A term logic for categorical programming. *Theor. Comp. Sci.*, 139:69–113, 1995.

[DF98] R. Diaconescu and K. Futatsugi. CAFEOBJ *Report: The Language, Proof Techniques and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

[DM94] O. De Moor. Categories, relations and dynamic programming. *Mathematical Structures in Computing Science*, 4:33–69, 1994.

[DMN68] O.-J. Dahl, B Myhrhang, and K. Nygaard. The SIMULA 67 Common Base Language. Tech. Report, Norvegian Computing Center, 1968.

[DRS95] R. Duke, G. Rose, and G. Smith. OBJECT-Z: A specification language advocated for the description of standards. *Computer Standards and Interfaces*, 17:511–533, 1995.

[EFH83] H. Ehrig, W. Fey, and H. Hansen. ACT ONE: An algebraic specification language with two levels of semantics. Technical report, TR 83-01, Tech. Univ. Berlin, 1983.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, 1985.

[ESS90] H.-D. Ehrich, A. Sernadas, and C. Sernadas. From data types to object types. *Jour. of Information Processing and Cybernetics*, 1/2(26):33—48, 1990.

[FE00] M. Fokkinga and R. Eshuis. Comparing refinements for failure and bisimulation semantics. Technical report, Faculty of Computing Science, Enschede, 2000.

[Fen96] C. Fencott. *Formal Methods for Concurrency*. International Thomson Computer Press, 1996.

[Fic97] C. Ficher. CSP-OZ: A combination of OBJECT-Z and CSP. In H. Bowman and J. Derrick, editors, *FMOODS'97 - Formal Methods for Open Object-Oriented Distributed Systems (volume 2)*. Chapman and Hall, 1997.

[FL97] J. Fiadeiro and A. Lopes. Semantics of architectural connectors. In *Proc. of TAPSOFT'97*, pages 505–519. Springer Lect. Notes Comp. Sci. (1214), 1997.

[Fok92a] M. M. Fokkinga. Calculate categorically! *Formal Aspects of Computing*, 4(4):673–692, 1992.

[Fok92b] M.M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, Dept INF, Enschede, The Netherlands, 1992.

[Fok94] M.M. Fokkinga. Monadic maps and folds for arbitrary datatypes. Memoranda Informatica 94-28, University of Twente, Junho 1994.

[Fok96] M. M. Fokkinga. Datatype laws without signatures. *Math. Struct. in Comp. Sci.*, 6:1–32, 1996.

[Fre91] P. J. Freyd. Algebraically complete categories. In A. et al Carboni, editor, *Proc. of the 1990 Como Category Theory Conference*, pages 95–104. Springer Lect. Notes Maths. (1488), 1991.

[Gay95] S. Gay. *Linear Types for Communicating Processes*. Ph.D. thesis, University of London, 1995.

[GD94] J. Goguen and R. Diaconescu. Towards an algebraic semantics for the object paradigm. In H. Ehrig and F. Orejas, editors, *Proc. Tenth Workshop on Abstract Data Types*, pages 1–29. Springer Lect. Notes Comp. Sci. (785), 1994.

[GGM76] A. Giarratana, F. Gimona, and U. Montanari. Observability concepts in abstract data specifications. In *Proc. Mathematical Foundations of Computer Science*. Springer Lect. Notes Comp. Sci. (45), 1976.

[GH93] J. Guttag and J Horning. LARCH*: Languages and Tools for Formal Specification*. Springer-Verlag, 1993.

[Gib93] J. Gibbons. Upwards and downwards accumulations on trees. In R. S. Bird, C. C. Morgan, and J. C. P. Woodcock, editors, *Mathematics of Program Construction*, pages 122–138. Springer Lect. Notes Comp. Sci. (669), 1993.

[Gib97] J. Gibbons. Conditionals in distributive categories. CMS-TR-97-01, School of Computing and Mathematical Sciences, Oxford Brookes University, 1997.

[GK96] S. Goldsack and S. Kent (eds). *Formal Methods and Object Technology*. FACT. Springer-Verlag FACT, 1996.

[GM87] J. Goguen and J. Meseguer. Unifying functional, object-oriented and relational programming with logic semantics. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT Press, 1987.

[GM00] J. Goguen and G. R. Malcolm. A hidden agenda. *Theor. Comp. Sci.*, 245(1):55–101, 2000.

[Gog91] J. Goguen. Types as theories. In G. Reed, A. Roscoe, A. William, and R. Wachter, editors, *Topology and Categories in Computer Science*, pages 357–390. Oxford University Press, 1991.

[Gog96] J. Goguen. Parametrised programming and software architectures. In *Symposium on Software Reusability*. IEEE, 1996.

[GP95] J. Groote and A. Ponse. The syntax and semantics of μCRL. In *Algebra of Communicating Processes*, pages 26–62. Springer-Verlag, 1995.

[GR83] A. Goldberg and D. Robson. *Smalltalk'80: the Language and Its Implementation*. Addison-Wesley, 1983.

[Gro70] A. Grothendieck. Catégories fibrées et descente (exposé vi). In A. Grothendieck, editor, *Revêtement Etales et Groupe Fondamental (SGA 1)*, pages 145–194. Springer Lect. Notes Maths. (224), 1970.

[GS93] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering (volume I)*. World Scientific Publishing Co., 1993.

[GS98] H. P. Gumm and T. Schroeder. Covarieties and complete covarieties. In *CMCS'98 - Workshop on Coalgebraic Methods in Computer Science, Lisbon*. ENTCS, volume 11, Elsevier, March 1998.

[GTW78] J. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology*, pages 80–149. Prentice-Hall International, 1978.

[GTWW77] J. Goguen, J. Thatcher, E. Wagner, and J. Wright. Initial algebra semantics and continuous algebras. *Jour. of the ACM*, 24(1):68–95, January 1977.

[Gur93] Y. Gurevich. Evloving algebras, an attempt to discovery semantics. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*. World Scientific, 1993.

[GWM$^+$96] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OJB. In J. Goguen and G. Malcolm, editors, *Software Engineering with* OJB*: Algebraic Specification in Practice*. Cambridge University Press, 1996.

[Hag87a] T. Hagino. *Category Theoretic Approach to Data Types*. Ph.D. thesis, tech. rep. ECS-LFCS-87-38, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, 1987.

[Hag87b] T. Hagino. A typed lambda calculus with categorical type constructors. In D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Category Theory and Computer Science*, pages 140–157. Springer Lect. Notes Comp. Sci. (283), 1987.

[Hen84] P. Henderson. me too: A language for software specification and model building. Preliminary Report, University of Stirling, 1984.

[Hen88] M. C. Hennessy. *Algebraic Theory of Processes*. Series in the Foundations of Computing. MIT Press, 1988.

[HHJT98] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about Java classes in object-oriented languages. In C. Hankin, editor, *European Symposium on Programming*, pages 105–121. Springer Lect. Notes Comp. Sci. (1381), 1998.

[HJ98] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Information & Computation*, 145:105–121, 1998.

[Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[Hoa85] C. A. R Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, 1985.

[Hoo96] P. F. Hoogendijk. *A generic theory of datatypes*. Ph.D. thesis, Department of Computing Science, Eindhoven University of Technology, 1996.

[HP95] M. Hofmann and B. Pierce. A unifying type-theoretic framework for objects. *Jour. Functional Programming*, 5(4):593–635, 1995.

[HPW92] P. Hudak, S. L. Peyton Jones, and P. Wadler. Report on the programming language Haskell, a non-strict purely-functional programming language, version 1.2. *SIGPLAN Notices*, 27(5), May 1992.

[HR95] U. Hensel and H. Reichel. Defining equations in terminal coalgebras. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification*, pages 307–318. Springer Lect. Notes Comp. Sci. (906), 1995.

[ISO88] ISO. Information processing systems - open systems interconnection - LOTOS - a formal description technique based on the temporal ordering of observation behaviour. ISO/TC97/SC21/N DI8807, 1988.

[Jac95] B. Jacobs. Mongruences and cofree coalgebras. In V.S. Alagar and M. Nivat, editors, *Algebraic Methodology and Software Technology (AMAST)*, pages 245–260. Springer Lect. Notes Comp. Sci. (936), 1995.

[Jac96a] B. Jacobs. Object-oriented hybrid systems of coalgebras plus monoid actions. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology (AMAST)*, pages 520–535. Springer Lect. Notes Comp. Sci. (1101), 1996.

[Jac96b] B. Jacobs. Objects and classes, co-algebraically. In C. Lengauer B. Freitag, C.B. Jones and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Academic Publishers, 1996.

[Jac97] B. Jacobs. Behaviour-refinement of coalgebraic specifications with coinductive correctness proofs. In *TAPSOFT'97: Theory and Practice of Software Development*, pages 787–802. Springer Lect. Notes Comp. Sci. (1214), 1997.

[Jac99a] B. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishers B. V. (North-Holland), 1999.

[Jac99b] B. Jacobs. The temporal logic of coalgebras via Galois algebras. Techn. rep. CSI-R9906, Comp. Sci. Inst., University of Nijmegen, 1999.

[Jac02] B. Jacobs. Exercises in coalgebraic specification. In R. Crole, R. Backhouse, and J. Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Constuction*, pages 236–280. Springer Lect. Notes Comp. Sci. (2297), 2002.

[JC94] B. Jay and J. Cockett. Shaply types and shape polymorphism. In D. Sannella, editor, *Programming Languages and Systems — ESOP'94*, pages 302–316. Springer Lect. Notes Comp. Sci. (788), 1994.

[Jeu93] J. Jeuring. *Theories for Algorithm Calculation*. Ph.D. thesis, Utrecht University, 1993.

[JJ97] P. Jansson and J. Jeuring. POLYP - a polytypic programming language extension. In *POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 470–482. ACM Press, 1997.

[Jon80] Cliff B. Jones. *Software Development — a Rigorous Approach*. Series in Computer Science. Prentice-Hall International, 1980.

[Jon83] Cliff B. Jones. Specification and design of (parallel) programs. In R. E. A. Mason (IFIP), editor, *Information Processing 83*, pages 321–332. Elsevier Science Publishers B. V. (North-Holland), 1983.

[Jon86] Cliff B. Jones. *Systematic Software Development Using* VDM. Series in Computer Science. Prentice-Hall International, 1986.

[Jon96] Cliff B. Jones. Accommodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, 8(2):105–122, 1996.

[JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[JS90] G. Jones and M. Sheeran. Circuit design in RUBY. In *Formal Methods for VLSI Design*. North-Holland, 1990.

[JSV96] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Camb. Phil. Soc.*, 119:447–468, 1996.

[Kar98] B. von Karger. Temporal algebra. *Math. Struct. in Comp. Sci.*, 8:277–320, 1998.

[Kat96] P. Katis. *Categories and Bicategories of Processes*. PhD thesis, University of Sydney, 1996.

[Kel82] G. M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 1982.

[Kie98a] R. B. Kieburtz. Codata and comonads in HASKELL. Unpublished manuscript, 1998.

[Kie98b] R. B. Kieburtz. Reactive functional programming. In David Gries and Willem-Paul de Roever, editors, *Programming Concepts and Methods (PROCOMET'98)*, pages 263–284. Chapman and Hall, Junho 1998.

[KL95] R. B. Kieburtz and J. Lewis. Programming with algebras. In *Advanced Functional Programming*, pages 267–307. Springer Lect. Notes Comp. Sci. (925), 1995.

[Knu65] D. E.. Knuth. On the translation of languages from left to right. *Information and Control*, pages 607–39, 1965.

[Koc72] A. Kock. Strong functors and monoidal monads. *Archiv für Mathematik*, 23:113–120, 1972.

[KSW97a] P. Katis, N. Sabadini, and R. F. C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997.

[KSW97b] P. Katis, N. Sabadini, and R. F. C. Walters. Span(Graph): A categorical algebra of transition systems. In M. Johnson, editor, *6th Int. Conf. Algebraic Methods and Software Technology (AMAST)*, pages 332–336, Sydney, December 1997. Springer Lect. Notes Comp. Sci. (1349).

[KSW00] P. Katis, N. Sabadini, and R. F. C. Walters. On the algebra of systems with feedback and boundary. *Rendiconti del Circolo Matematico di Palermo*, II(63):123–156, 2000.

[KSWW01] P. Katis, N. Sabadini, R. F. C. Walters, and H. Weld. Categories of circuits. submitted for publication (Available from `www.unico.it/˜walters/`), 2001.

[Kur98] A. Kurz. Specifying coalgebras with modal logic. In *CMCS'98 - Workshop on Coalgebraic Methods in Computer Science, Lisbon*. volume 11 of ENTCS, March 1998.

[Kur01] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. Ph.D. Thesis, Fakultat fur Mathematik, Ludwig-Maximilians Univ., Muenchen, 2001.

[KW92] D. King and P. Wadler. Combining monads. In *Proc. 5th Annual Glasgow Workshop on Functional Programming*, 1992.

[Len98] M. Lenisa. *Themes in Final Semantics*. PhD thesis, Universita de Pisa-Udine, 1998.

[LMH98] D. Leijen, E. Meijer, and J. Hook. HASKELL as an automated controller. In S. D. Swierstra, P. R. Henriques, and J. N. Oliveira, editors, *Third International Summer School on Advanced Functional Programming, Braga*, pages 268–289. Springer Lect. Notes Comp. Sci. (1608), September 1998.

[LS97] F. W. Lawvere and S. H. Schanuel. *Conceptual Mathematics*. Cambridge University Press, 1997.

[MA86] E. Manes and A. Arbib. *Algebraic Approaches to Program Semantics*. Texts and Monographs in Computer Science. Springer Verlag, 1986.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*. Springer Verlag, 1971.

[Mal90a] G. R. Malcolm. *Algebraic data types and program transformation*. Ph.D. thesis, Department of Computing Science, Groningen University, The Netherlands, 1990.

[Mal90b] G. R. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14(2–3):255–279, 1990.

[Mal96] G. Malcolm. Behavioural equivalence, bisimulation and minimal realization. In O. Owe and O.-J. Dahl, editors, *Proc. Recent Trends in Data Type Specification*, pages 359–378. Springer Lect. Notes Comp. Sci. (1130), 1996.

[Man98] E. Manes. Implementing collection classes with monads. *Math. Struct. in Comp. Sci.*, 8:231–276, 1998.

[McC60] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Comm. ACM*, 3(4):184–195, 1960.

[McC63] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirshberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, 1963.

[McL92] C. McLarty. *Elementary Categories, Elementary Toposes*, volume 21 of *Oxford Logic Guides*. Clarendon Press, 1992.

[MDEK95] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *5th European Software Engineering Conference*, 1995.

[Mea55] G. H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Techn. Jour.*, 34(5):1045–1079, 1955.

[Mee92] L. Meertens. Paramorphisms. *Formal Aspects of Computing*, 4(5):413–425, 1992.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theor. Comp. Sci.*, 96(1):73–155, 1992.

[Mes00] J. Meseguer. Rewriting logic and Maude: A wide-spectrum semantic framework for object-based distributed systems (invited lecture). In S. F. Smith and C. L. Talcott, editors, *FMOODS'2000 - Formal Methods for Open Object-Oriented Distributed Systems*, pages 89–117. Kluwer Academic Publishers, September 2000.

[Mey88] B. Meyer. *Object-Oriented Software Construction*. Series in Computer Science. Prentice-Hall International, 1988.

[MFP91] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In J. Hughes, editor, *Proceedings of the 1991 ACM Conference on Functional Programming Languages and Computer Architecture*, pages 124–144. Springer Lect. Notes Comp. Sci. (523), 1991.

[Mic92] Microsoft. *The* COM *Reference*. Microsoft Press, 1992.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer Lect. Notes Comp. Sci. (92), 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice-Hall International, 1989.

[Mil99] R. Milner. *Communicating and Mobile Processes: the π-Calculus*. Cambridge University Press, 1999.

[MJ95] E. Meijer and J. Jeuring. Merging monads and folds for functional programming. In J. Jeuring and E. Meijer, editors, *International Summer School on Advanced Functional Programming*, pages 228–266. Springer Lect. Notes Comp. Sci. (925), 1995.

[MM99] B. Meyer and C. Mingins. Component-based development: From buzz to spark. *IEEE Computer*, 32(7):35–37, 1999.

[Mog89] E Moggi. Computational lambda-calculus and monads. In *Proceedings of the Logic in Computer Science Conference*, 1989.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[Mon00] L. Monteiro. Observation systems. In H. Reichel, editor, *CMCS'00 - Workshop on Coalgebraic Methods in Computer Science*. ENTCS, volume 33, Elsevier, 2000.

[Moo66] E. F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*, pages 129–153. Princeton University Press, 1966.

[Mos99] L. Moss. Coalgebraic logic. *Ann. Pure & Appl. Logic*, 1999.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1992.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.

[MQ94] M. Moriconi and X. Qian. Correctness and composition of softare architectures. In *Second Symposium on Foundations of Software Engineering*, pages 164–174. ACM Press, 1994.

[ND95] O. Nierstrasz and L. Dami. Component-oriented software technology. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, pages 3–28. Prentice-Hall International, 1995.

[Nie93] O. Nierstrasz. Regular types for active objects. In *OOPSLA'93*, pages 1–15. volume 28 of ACM Sigplan Notices, 1993.

[Oli84] J. N. Oliveira. *The Formal Semantics of Deterministic Dataflow Programs*. PhD thesis, Department of Computer Science, University of Manchester, February 1984.

[Oli90] J. N. Oliveira. A reification calculus for model-oriented software specification. *Formal Aspects of Computing*, 2(1):1–23, 1990.

[Oli91] J. N. Oliveira. *Especificação Formal de Programas*. University of Minho, 1^{th} edition, 1991. Lecture Notes for M.Sc. Course in Computing (in Portuguese, last edition 1994).

[Oli92a] J. N. Oliveira. Formal Software Development. Lecture Notes for the MSc in Computer Science, Minho University, 1992.

[Oli92b] J. N. Oliveira. Software reification using the SETS calculus. In *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992. (Invited paper).

[Oli93] J. N. Oliveira. The CAMILA strategy for software reusability. *ERCIM News*, 14:13–14, July 1993.

[Oli97] J. N. Oliveira. Can distribution be (statically) calculated? Technical report, UNU/IIST, Macau, May 1997.

[Oli98] J. N. Oliveira. A data structuring calculus and its application to program development, 1998. Lecture Notes of M.Sc. Course (150 p.). Maestria em Ingeneria del Software, Departamento de Informatica, Facultad de Ciencias Fisico-Matematicas y Naturales, Universidad de San Luis, Argentina.

[Par72] D. Parnas. Information distribution aspects of design methodology. In *Information Processing '72*, pages 339–344. North-Holland, 1972.

[Par81] D. Park. Concurrency and automata on infinite sequences. pages 561–572. Springer Lect. Notes Comp. Sci. (104), 1981.

[Par96] A. Pardo. A calculational approach to strong datatypes. In *Selected Papers from 8th Nordic Workshop on Programming Theory*. Research Report 240, Oslo, 1996.

[Par98] A. Pardo. Monadic corecursion: Definition, fusion laws and applications. In *CMCS'98 - Workshop on Coalgebraic Methods in Computer Science, Lisbon*. volume 11 of ENTCS, March 1998.

[Par00] A. Pardo. Towards merging recursion and comonads. In *WGP'2000 Workshop in Generic Programming, Ponte de Lima*. Utrecht University Tech. Rep. UU-CS-2000-19, July 2000.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, 1962.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symp. on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[Pra95] V.R. Pratt. Chu spaces and their interpretation as concurrent objects. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, pages 392–405. Springer Lect. Notes Comp. Sci. (1000), 1995.

[PW98] J. Power and H. Watanabe. An axiomatics for categories of coalgebras. In *CMCS'98 - Workshop on Coalgebraic Methods in Computer Science, Lisbon*. ENTCS, volume 11, Elsevier, March 1998.

[RE98] W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*, volume 47 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[Rei81] H. Reichel. Behavioural equivalence —a unifying concept for initial and final specifications. In *Third Hungarian Computer Science Conference*. Akademiai Kiado, Budapest, 1981.

[Rei85] W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[Rei88] W. Reisig. Temporal logic and causality in concurrent systems. In F. H. Vogt, editor, *Proc. CONCUR' 88*. Springer Lect. Notes Comp. Sci. (335), 1988.

[Rei91] W. Reisig. Petri nets and algebraic specifications. *Theor. Comp. Sci.*, 80(1):1–34, 1991.

[Rei95] H. Reichel. An approach to object semantics based on terminal co-algebras. *Math. Struct. in Comp. Sci.*, 5:129–152, 1995.

[RJT01] J. Rothe, B. Jacobs, and H. Tews. The coalgebraic class specification language CCSL. *Jour. of Universal Computer Science*, 7(2), 2001.

[Ros00] G. Rosu. *Hidden Logic*. Ph.D. thesis, University of California, San Diego, 2000.

[RS92] L. Rapanotti and A. Socorro. Introducing FOOPS. Techn. Report PRG-TR-28-92, Computing Laboratory, Oxford, UK, 1992.

[RT94] J. Rutten and D. Turi. Initial algebra and final co-algebra semantics for concurrency. In *Proc. REX School: A Decade of Concurrency*, pages 530–582. Springer Lect. Notes Comp. Sci. (803), 1994.

[Rut95] J. Rutten. A calculus of transition systems (towards universal co-algebra). In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra, A Bisimulation Perspective*, CSLI Lecture Notes (53), pages 231–256. CSLI Publications, Stanford, 1995.

[Rut96] J. Rutten. Universal coalgebra: A theory of systems. Technical report, CWI, Amsterdam, 1996.

[Rut98] J. Rutten. Automata and coinduction (an exercise in coalgebra). In *Proc. CONCUR' 98*, pages 194–218. Springer Lect. Notes Comp. Sci. (1466), 1998.

[Rut00] J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249(1):3–80, 2000. (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[Rut01] J. Rutten. Elements of stream calculus (an extensive exercise in coinduction). Technical report, CWI, Amsterdam, 2001.

[Sch97] M. A. Schroeder. Higher-order Charity. Master's thesis, The University of Calgary, 1997.

[Sch98] D. Schamschurko. Modeling process calculi with PVS. In *CMCS'98 - Workshop on Coalgebraic Methods in Computer Science, Lisbon*. ENTCS, volume 11, Elsevier, March 1998.

[See89] R. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In J. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic, Contemporary Mathematics, volume 92*, pages 371–382. American Mathematical Society, 1989.

[Sel99] P. Selinger. Categorical structure of asynchrony. In *MFPS'98 (invited talk), New Orleans*. ENTCS, volume 20, Elsevier, March 1999.

[SFSE89] A. Sernadas, J. Fiadeiro, C. Sernadas, and H.-D. Ehrich. Abstract object types: A temporal perspective. In A. Pnueli, H. Barringer, and B. Banieqbal, editors, *Proc. Colloquium on Temporal Logic and Specification*. LNCS (398), 1989.

[She93] T. Sheard. Type parametric programming. Technical report, Oregon Graduate Institute of Science and Technology, Portland, USA, 1993.

[SN99] J.-G. Schneider and O. Nierstrasz. Components, scripts, glue. In L. Barroca, J. Hall, and P. Hall, editors, *Software Architectures - Advances and Applications*, pages 13–25. Springer-Verlag, 1999.

[SP82] M. Smyth and G. Plotkin. The category theoretic solution of recursive domain equations. *SIAM Journ. Comput.*, 4(11):761–783, 1982.

[Spe93] D. L. Spencer. *Categorical Programming with Functorial Strength*. PhD thesis, The Oregon Graduate Institute of Science and Technology, Janeiro 1993.

[Spi92] J. M. Spivey. *The* Z *Notation: A Reference Manual (2nd ed)*. Series in Computer Science. Prentice-Hall International, 1992.

[Spo97] D. Spooner. *Building Process Categories*. Ph.D. thesis, University of Calgary, Alberta, 1997.

[ST85] D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. In *CAAP 85*, pages 413–427. Springer Lect. Notes Comp. Sci. (185), 1985.

[Sti92] C. Stirling. Modal and temporal logics. In Maibaum Abramsky, Gabbay, editor, *Handbook of Logic in Computer Science (vol. 2)*, pages 478–551. Oxford Science Publications, 1992.

[Sti95] C. Stirling. Modal and temporal logics for processes. *Springer Lect. Notes Comp. Sci. (715)*, pages 149–237, 1995.

[Str96] R. Street. Categorical structures. In M. Hazewinkel, editor, *Handbook of Algebra (vol. 1)*, pages 529–577. Elsevier North-Holland, 1996.

[Str99] Th. Streicher. Fibred categories. Lecture notes, Spring School on Categorical Methods in Logic and Computer Science, LMU, Muenchen, April 1999.

[Szy98] C. Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[Tar55] A. Tarski. A lattice–theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[TP97] D. Turi and G.D. Plotkin. Towards a mathematical operational semantics. In *Proc. 12*$^{\text{th}}$ *LICS Conf.*, pages 280–291. IEEE, Computer Society Press, 1997.

[TR98] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Math. Struct. in Comp. Sci.*, 8(5):481–540, 1998.

[Tur95] D. Turner. Elementary strong functional programming. In *Proc. Inter. Sym. on Functional Programming Languages in Education*, pages 1–13. Springer Lect. Notes Comp. Sci. (1022), 1995.

[Tur96] D. Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University of Amsterdam, June 1996.

[UV99] T. Uustalu and V. Vene. Primitive (co)recursion and course-of-values (co)iteration, categorically. *INFORMATICA (IMI, Lithuania)*, 10(1):5–26, 1999.

[UVP01] T. Uustalu, V. Vene, and A. Pardo. Recursion schemes from comonads. *Nordic Journal of Computing (to appear)*, 2001.

[Val00] J. Valença. Reactive systems and dependent types. LOGCOMP Final Workshop, 2000.

[Veg97] S. Veglioni. *Integrating Static and Dynamic Aspects in the Specification of Open Object-based Distributed Systems*. PhD thesis, Oxford University Computing Laboratory, 1997.

[vG90] R. van Glabbeek. The linear time - branching time spectrum. In J. Baeten and J. Klop, editors, *Proc. CONCUR '90*. Springer Lect. Notes Comp. Sci. (458), 1990.

[VU97] V. Vene and T. Uustalu. Functional programming with apomorphisms (corecursion). In *Proc. 9th Nordic Workshop on Programming Theory*, 1997.

[Wad89] P. Wadler. Theorems for free! In *4th International Symposium on Functional Programming Languages and Computer Architecture*, pages 347–359, September 1989.

[Wad92] P. Wadler. Comprehending monads. *Math. Struct. in Comp. Sci.*, 2:461–493, 1992. (Special issue of selected papers from 6'th Conference on Lisp and Functional Programming.).

[Wad95] P. Wadler. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*. Springer Lect. Notes Comp. Sci. (925), 1995.

[Wal89] R. F. C. Walters. Datatypes in distributive categories. *Bull. of the Australian Mathematical Society*, 40:79–82, 1989.

[Wal91] R. F. C. Walters. *Categories and Computer Science*, volume 28 of *Cambridge Computer Science Texts*. Cambridge University Press, 1991.

[Wan79] M. Wand. Final algebraic semantics and data type extensions. *Jour. Comput. Systems Sci.*, 19:27–44, 1979.

[WD96] J. Woodcock and J. Davies. *Using* Z*: Specification, Refinement and Proof.* Prentice-Hall International, 1996.

[Wel98] H. Weld. *On Categories of Asynchronous Circuits*. PhD thesis, University of Sydney, 1998.

[WF98] M. Wermelinger and J. Fiadeiro. Connectors for mobile programs. *IEEE Trans. on Software Eng.*, 24(5):331–341, 1998.

[Wir90] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (volume B)*, pages 673–788. Elsevier - MIT Press, 1990.

[WLF01] M. Wermelinger, A. Lopes, and J. Fiadeiro. A graph based architectural (re)configuration language. In *Proc. of ESEC/FSE'01*. ACM Press (in print), 2001.

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Gabbay, editors, *Handbook of Logic in Computer Science (vol. 4)*, pages 1–148. Oxford Science Publications, 1995.

[Wol99] U. Wolter. A coalgebraic introduction to CSP. In *CMCS'99 - Workshop on Coalgebraic Methods in Computer Science*. ENTCS, volume 19, Elsevier, 1999.

[Wol00] U. Wolter. On corelations, cokernels and coequations. In H. Reichel, editor, *CMCS'00 - Workshop on Coalgebraic Methods in Computer Science*, pages 347–366. ENTCS, volume 33, Elsevier, 2000.

[Wor98] J. Worrell. A topos of hidden algebras. In *CMCS'98 - Workshop on Coalgebraic Methods in Computer Science, Lisbon*. ENTCS, volume 11, Elsevier, March 1998.

[Wra88] G. C. Wraith. A note on categorical data types. In D. et all Pitt, editor, *Proc. Category Theory and Computer Science*, pages 118–127. Springer Lect. Notes Comp. Sci. (389), 1988.

[WW99] P. Wadler and K. Weihe. Component-based programming under different paradigms. Technical report, Report on the Dagstuhl Seminar 99081, February 1999.