# Structuring structural operational semantics

# Structuring Structural Operational Semantics

## PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 26 september om 16.00 uur

door

MohammadReza Mousavi

geboren te Teheran, Iran

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.F. Groote
en
prof.dr. G.D. Plotkin

Copromotor:
dr.ir. M.A. Reniers

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Mousavi, MohammadReza

Structuring Structural Operational Semantics / MohammadReza Mousavi. -
Eindhoven : Technische Universiteit Eindhoven, 2005.
Proefschrift. - ISBN 90-386-0644-3
NUR 993
Subject headings: programming ; formal methods / programming languages ; semantics
CR Subject Classification: F.3.2

Eerste promotor: prof.dr.ir. J.F. Groote (Technische Universiteit Eindhoven)

Tweede promotor: prof.dr. G.D. Plotkin (University of Edinburgh)

Copromotor: dr.ir. M.A. Reniers (Technische Universiteit Eindhoven)

Kerncommissie:
prof.dr. L. Aceto (Aalborg Universitet and Reykjavíc University)
prof.dr. J.C.M. Baeten (Technische Universiteit Eindhoven)
prof.dr. W.J. Fokkink (Vrije Universiteit Amsterdam)

The work in this thesis is supported by NWO as a part of project SACC (612.063.001).

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

IPA dissertation series 2005-15



Printing: Eindhoven University Press

Cover design: S.E. Baha

Illustration front cover: M.C. Escher's "Print Gallery (Rework by Lenstra and de Smit)" © 2005 The M.C. Escher Company B.V. - Baarn - Holland. All rights reserved.
For more on "Escher and the Droste Effect" see `http://escherdroste.math.leidenuniv.nl/` .

# Contents

# Chapter 0

# Preface

> “Everything that is really great and inspiring is created by the individual who can labor in freedom.”
>
> [Albert Einstein]

Four years ago, when I was leaving Iran to start my Ph.D. studies, the then supervisor of mine gave me some pieces of advice and told me about his experiences with studying abroad. Among those, he warned me that I had better be prepared for a kind of cultural shock in the coming couple of months. He advised me to make myself busy with all kind of activities and sports to get through it. To my surprise, his prediction did not come true. I found myself happy and content with my new situation and did not feel the rather drastic social and cultural change in my surroundings. Part of this smooth transition must have been due to my sheer enthusiasm in continuing my studies, but for a great deal, I owe this to the kind and tolerant people who created a wonderful working environment around me. I devote this preface to thanking all those who helped me start this enjoyable path and finish it conveniently.

My thanks goes to Michel Chaudron for accepting me in his project and for his friendship throughout these four years.

Michel Reniers was my daily supervisor and a member of my Ph.D. project. Jan Friso Groote (my thesis supervisor, to be acknowledged shortly) described him once as a "*locomotive*" and he is truly so; he has enormous power and enthusiasm in his work and he pulls those connected to him with this huge power. He has always been there when I needed his help. More importantly, he taught me to perform independent research with a carefully chosen level of supervision. Without his help, I could not have been at this point and hence, I express my best thanks to him for his help and supervision.

Twan Basten was the other member of my Ph.D. project. He is one of the most careful and eager reviewers of scientific papers, I have seen in my life. His insightful comments have been crucial in the publications I co-authored with him. Above all, his enthusiasm and trust in my work has always given me more confidence in what I have been doing. For all that and more, I wholeheartedly thank him.

Jan Friso Groote has been my thesis supervisor and our group leader. I highly appreciate working with him and under his supervision and I gratefully thank him for his supervision, friendship and "Sunday noon-time chats". He has been too friendly and kind to be just a boss and too influential and experienced to be just a friend.

Jan Friso, Michel and Twan were so kind to create a joint position for me to stay in Eindhoven. I highly appreciate their effort and I am looking forward to more and more fruitful cooperations with them.

The members of my thesis committee are gratefully acknowledged for reading the thesis, providing useful comments and being present in my defense session. It was my privilege to have Luca Aceto, Jos Baeten, Wan Fokkink, Jan Friso Groote, Gordon Plotkin and Michel Reniers in the kernel thesis committee and Mark de Berg and Michel Chaudron and First Vaandrager in the defense opposition. In particular, I would like to thank Gordon Plotkin who kindly accepted to be my

thesis co-supervisor.

In the past four years, I had the opportunity to cooperate with many people and several groups from different institutes. For these opportunities, I am obliged to Sandeep Shukla from Virginia Tech., Blacksburg, USA, Jean-Pierre Talpin, Paul Le Guernic and Jean-Pierre Banâtre from INRIA/IRISA, Rennes, France, Farhad Arbab from CWI, Amsterdam, The Netherlands, Marjan Sirjani from Sharif University of Technology, Tehran, Iran and Jamie Gabbay from King's College London, London, UK. Chapter 9 is the result of cooperation with Jamie and initial ideas for Chapter 7 emerged in the joint work with Farhad and Marjan. I also acknowledge the funding provided by Espresso and Paris projects for my second visit to INRIA/IRISA. David Berner and Abdoulaye Gamatié were great friends and helped me out with various things during my visits to France. Gordon Plotkin, John Power and Alex Simpson were extremely hospitable and kind during my short visit to University of Edinburgh. Although I could not fit all the results of cooperations with these good colleagues in this thesis, they have certainly influenced the state of my mind and hence they are indirectly present in this thesis.

I thank all the members of OAS group for being very good friends, for organizing enjoyable "groepsuitjes", for listening to my boring stories in OAS colloquia and giving good comments on them as well as on the drafts of my reports. My thanks go to Peter van den Brand, Pieter Cuijpers, Mugur Ionita, Nicu Goga, Rob Hoogerwoord, Adam Koprowski, Arjan Mooij, Jaco van de Pol, Judi Romijn, Olga Tveretina, Muck van Weerdenburg, Wieger Wesselink and Hans Zantema. I have also had a very nice time with my good friends and colleagues at EESI and SAN, FM and VIS groups. Particularly, I thank Giovanni Russello, Kalok Man, Jinfang Huang, Fei Zuo, Ana Sokolova, Tijn Borghuis, Dragan Bosnacki, Louis van Gool, Gueorgui Jojgov, Uzma Khadim, Bas Luttik, Ronald Middeldorp, Erik de Vink, Tim Willemse, Frank van Ham, Arjan Kok, Elisabeth Melby and Hannes Pretorius for their friendship. Giovanni, Pieter, Kalok, Peter, Muck and Adam deserve another special thank for tolerating me as an officemate.

Tineke van den Bosch has been so kind to arrange every single administrative issue from the date I arrived in Eindhoven for my job interview to the date. Monique Bechtold was also very kind and helpful during the dark ages when I did not understand a word of Dutch and had a dozen official Dutch letters in my mailbox. I thank them both for their help.

The valuable friendship I had with my neighbors in the student housing of TU/e has continued, in most cases, to date. I enjoyed the company of Lidia Sandra, Harry Tanjung and Ahmad Reslan (who are back to their homelands) and highly appreciate the continuing company of Abdool Saib, Rabah Hanfough and Nathalia Romero Herrera. Abdool kept on being a neighbor when I moved out of student housing and hence I am tempted to conclude that he has been enjoying it; in my case, it has been certainly so!

The circle of Iranian friends at TU/e has always given me a refuge to get a taste of home. It all started with Ehsan Baha and his football team and continued with the kind company of Hamed and Negar Fatemi, AmirHossein Ghamarian, MohammadAli Abam, Kamyar Malakpoor, Mohammad Frashi, Saeid Talebi and many other good friends. AmirHossein and MohammadAli did a great job as the mayors of our shared house. Majid Nili (now in Cincinnati, OH) remained a chum while being thousands of miles away. Mortaza Bargh has been a very kind and helpful friend since the early days of my arrival in the Netherlands. May the hands of you all never pain (giving thanks, the Iranian way).

Managers and secretaries of Embedded Systems Institute (formerly, Eindhoven Embedded System Institute), were very kind and hospitable and warmly received me in their community during the first two years of my studies. My appreciations goes to: Marloes van Lierop, Lia de Jong, Miranda Willems, Franka van Neerven, Martin Rem and Anget Mestrom for creating such a nice working environment.

My supervisors and professors at Sharif University of Technology introduced me to the wonderful worlds of academic research, Computer Science and Formal Methods. For that, my best thanks goes to Seyyed-Hasan Mirian (my ex-supervisor), Mohammad Ardeshir, Ali Movaghar and Rasool Jalili.

The teachers of the language center at TU/e were highly influential in my getting acquainted with the Dutch language and culture. It was due to the kind effort of Nelleke de Vries, Elly Arkesteijn and Pieter uit den Boogaart that I could read “The Undutchables” in Dutch! Hartelijk dank allemaal.

In Eindhoven, finding a reasonable place to live in is a grand challenge of its own. My thanks go to Lettie Werkman, Martijn Willemsen and Ellen Melis for helping me solve this challenge in different stages.

The last but certainly not the least comes my family. Words cannot express the extent to which I feel indebted and grateful to them for all their unconditional help and support throughout my whole life and in particular, during the last four years. Hence, *with love and gratitude, I dedicate this thesis to them.*

# Chapter 1

# Introduction

"Allez en avant, ... et la foi vous viendra."
(Just go on, ... and faith will catch up with you.)

[Jean d'Alembert]

## 1.1 The Subject Matter

> In 1897, Michel Bréal coined the word "*semantics*" in a book that revolutionized our approach to the study of language [34].

"Semantics" was originally meant to study the evolution of meaning in languages. But the meaning of the word itself has evolved and is now used in linguistics to denote "the science of meaning" in its broad sense. Semantics is often used in contrast with syntax which refers to the structure, rather than the meaning, of the language under study. In Computer Science, semantics has essentially the same meaning. Only here, the languages under study are artificial computer-related languages rather than natural ones used in human communication. As a result, the semantics of such languages are synthetic; They have to be defined by the developers of such languages rather than being inferred from their practical usage.

Computer languages have a simpler structure and are meant to be less ambiguous than natural languages. Reducing ambiguity in the semantics of computer-related languages is among the first steps towards developing rigorous methods of reasoning about computer systems. Mathematics is a useful means to this end; by associating a *formal* (mathematical) semantics to computer languages we are able to disambiguate them, thanks to the inherent precision and clarity in mathematics. New computer languages appear frequently in different fields of Computer Science and existing languages are constantly extended with new features. Hence, developing methods for defining formal semantics and providing meta-theories for reasoning about the semantics are of an overwhelming importance and can be beneficial for a large community of computer scientists.

Within the field of formal semantics, there are different flavors of associating meanings to programs. Two mainstream examples of such flavors are *denotational* and *operational semantics* (cf. [136] for a general overview of the field). The denotational method is aimed at defining a function (denotation) which associates semantic objects to pieces of syntax. The definition of this function is often given recursively by a structural induction on syntactic constructs. The operational method, on the other hand, defines a transition relation among syntactic objects representing the execution of programs on an abstract machine. Although the denotational method has its own advantages (e.g., being inherently compositional), the operational approach has been widely accepted among the language developers and practitioners since it is easy to understand and close to implementation.

*Structural Operational Semantics (SOS)* [66, 105, 107, 108] was introduced by Gordon Plotkin in [106] as a logical means to defining operational semantics. The basic idea behind SOS is to define the behavior of a program in terms of the behavior of its parts, thus providing a structural, i.e., syntax oriented, view on operational semantics. Thanks to its intuitive look and easy to follow structure, SOS has gained great popularity and has become a de facto standard in defining operational semantics. As a sign of success, the original report (so-called Aarhus report) on SOS [106] has attracted some 900 citations according to the CiteSeer search engine to date!

This enormous popularity and vast application of SOS has called (and still calls) for more theoretical work. Many researchers have responded to this call and have spent huge effort on laying firm mathematical foundations for different aspects of SOS. We provide a non-conclusive overview of these works in Chapter 3 of this thesis. This thesis also reports a number of attempts to improve on some of the existing mathematical frameworks and meta-results about SOS. These improvements are achieved by adding more syntactic features and structures to the traditional SOS formats and suggesting ways to prove useful semantic properties based on the structure of SOS specifications.

## 1.2 Contributions

A Ph.D. thesis is traditionally meant to put a clear "thesis" forward and justify in the course of the discussions. If I am to formulate such a thesis, I would phrase it as *the possibility of establishing a general yet structured framework for Structural Operational Semantics*. Being a pupil of the formal methods' school, I am tempted to blame the ambiguity of natural language for the vagueness of the above thesis and start writing in Greek letters to explain it formally. But for once, I defer writing in Greek till the coming chapters and try to explain the phrases in my thesis in natural language in the remainder.

- *possibility*: possibility by itself should not be very interesting, but rather a constructive proof of possibility is sought. In other words, I am aiming at giving concrete instances of such frameworks throughout my Ph.D. thesis.

- *general*: By general, I mean that the theory should be able to deal with the semantics of more languages, be it existing ones or those that are yet to be devised in the future. It is hard, if not impossible, to foresee all such future instance and in some cases, I was not even able to deal with all existing ones but generality remains a goal in my endeavor;

- *structured*: Following [64], structured, in my terminology, means the possibility of inferring some intuitive and precisely defined properties from the semantic framework without delving into details of an instance each time;

- *framework*: a framework consists of a syntactic structure for SOS specifications and a method for inferring a behavioral model from the specification; throughout this thesis, we only extend the syntactic part of the SOS framework and reuse existing methods of inferring behavioral models;

- *Structural Operational Semantics*: SOS is the subject matter of this thesis which was briefly introduced in Section 1.1 and is explained in detail in Chapter 2.

One may argue that my thesis has already been proven by the existing SOS frameworks (e.g., by those reported in [5]). However, the existing frameworks were neither the most general nor the most structured of all. To show this, throughout this thesis, I add more generality and structure and make the following contributions to them.

- Proposing meta-theorems to prove commutativity of certain operators by examining the structure of SOS specifications [Chapter 4];
- Interpreting structural congruences as equational addenda to SOS; extending congruence and well-definedness results to SOS specifications with structural congruences [Chapter 5];
- Suggesting a more liberal notion of conservative extension, called orthogonality, which allows for equality-preserving addition of behavior to the old language; presenting and proving meta-theorems about orthogonality [Chapter 6];
- Implementing a prototyping environment for SOS specifications in the Maude rewriting language [Chapter 7];
- Extending the existing SOS frameworks to accommodate data as part of the state (thus, catering for entities such as storage and memory, timing, valuation of continuous model variables, etc., in the operational state); studying notions of equivalence with data and introducing rule formats to make these equivalences a congruence, i.e., compositional [Chapter 8];
- Extending the SOS framework in order to make it appropriate for semantic specification of higher order processes; formulating congruence meta-theorem for strong as well as higher order bisimilarity [Chapter 9].

## 1.3 Suggested Method of Reading

Chapter 2 gives a basic introduction to SOS meta-theory and will define the generic formalization of SOS in terms of *Transition System Specifications (TSS)*. It will also define how a TSS may induce a (labelled) transition relation among operational states. Hence, Chapter 2 will serve as a basis for the rest of the thesis. Note that the foundation laid down in this chapter is quite general and thus, in the rest of the thesis, we usually simplify it and deal with more restricted and workable cases.

Chapter 3 presents an overview of the existing results about SOS frameworks. Some of these results form the basis for the improvements proposed in the rest of the thesis. However, we explicitly mention the particular results to be recalled in each chapter so that one may refer to Chapter 3 only when needed.

Chapters 4 to 9 are independent from each other and, besides basic definitions that have to be recalled from Chapters 2 and 3, can be read on their own.

# Chapter 2

# Structural Operational Semantics

> "Development of Western Science is based on two great achievements - the invention of the formal logical system (in Euclidean geometry) by the Greek philosophers, and the discovery of the possibility to find out causal relationships by systematic experiment (during the Renaissance). In my opinion, one has not to be astonished that the Chinese sages have not made these steps. The astonishing thing is that these discoveries were made at all."
>
> [(Attributed to) Albert Einstein]

> "You may always depend on it that algebra, which cannot be translated into good English and sound common sense, is bad algebra."
>
> [William Kingdon Clifford]

> "I saw the [SOS] rules as directly formalising the natural English description ..."
>
> [Gordon D. Plotkin [107]]

## 2.1 Introduction

Structural Operational Semantics is a logical way to define operational semantics. Operational semantics defines the possible *transitions* that a piece of syntax can make during its "execution" on an abstract machine. Each transition may be *labelled* by a message to be communicated to the outside world. Transitions of a composite piece of syntax can usually be defined, in a generic way, in terms of the transitions of its constituting parts. This forms the central idea behind Structural Operational Semantics.

To give an impression about SOS specifications, we specify the operational semantics of a simple programming language in this style. Several examples, including ones similar to this programming language, will be treated formally throughout this thesis and their properties will be investigated.

**Example 2.1** The first example is a simple programming language with the syntax specified below.

$$\begin{array}{lcl}
\mathit{Prg} & ::= & \mathtt{skip} \mid \mathit{Assign} \mid \mathtt{if}\ (\mathit{BExp})\ \mathtt{then}\ \mathit{Prg}\ \mathtt{else}\ \mathit{Prg}\ \mathtt{fi} \mid \mathit{Prg}; \mathit{Prg} \\
\mathit{BExp} & ::= & \top \mid \bot \mid \mathit{Chk} \mid (\mathit{BExp} \vee \mathit{BExp}) \mid (\neg \mathit{BExp}) \\
\mathit{Assign} & ::= & \mathit{Name} := \mathit{Val} \\
\mathit{Chk} & ::= & \mathit{Name} == \mathit{Val}
\end{array}$$

The syntax consists of a constant for the terminated program `skip`, assignment $\mathit{Name} := \mathit{Val}$ of a value $\mathit{Val}$ to a variable named $\mathit{Name}$, the conditional $\mathtt{if}\_\ \mathtt{then}\_\ \mathtt{else}\_\ \mathtt{fi}$ statement and the sequential composition of two programs $\_;\_$.

The semantics for the evaluation of the boolean expressions is given next. The state of this semantics is $[\mathit{BExp},\ \mathit{Mem}]$ where $\mathit{BExp}$ is a boolean expression and $\mathit{Mem}$ is a representation of the memory with the following syntax.

$$\mathit{Mem} \quad ::= \quad \mathtt{nil} \mid (\mathit{Name} \mapsto \mathit{Val}) +\!\!+\, \mathit{Mem}$$

A memory is a list of memory cells, each assigning a value to a variable name. The list of memory cells ends with `nil`. Without making it explicit, we assume that all variables mentioned in a program have exactly one corresponding cell in the memory of the operational state.

$$\frac{}{Holds([\top,\ M])} \qquad \frac{}{Holds([n == v,\ (n \mapsto v) +\!\!+ M])}$$

$$\frac{n \neq n' \quad Holds(\langle n == v,\ M\rangle)}{Holds([n == v,\ (n' \mapsto v') +\!\!+ M])}$$

$$\frac{\neg Holds([b,\ M])}{Holds([\neg b,\ M])}$$

$$\frac{Holds([b_0,\ M])}{Holds([b_0 \vee b_1,\ M])} \qquad \frac{Holds([b_1,\ M])}{Holds([b_0 \vee b_1,\ M])}$$

The above semantics of boolean expressions, defines a predicate *Holds* on the operational state of boolean expressions. The deduction rules should be read as: the predicate in the conclusion (below the vertical line) holds if the premises (statements above the line, if any) are valid. Based on the semantics of the boolean expressions, given above, the operational semantics of a program is defined as follows. The state of this semantics is of the form $\langle Prg,\ Mem\rangle$ where *Prg* is a program and *Mem* is a memory.

$$\frac{}{\langle n := v,\ (n \mapsto v') +\!\!+ M\rangle \to \langle \texttt{skip},\ (n \mapsto v) +\!\!+ M\rangle}$$

$$\frac{n \neq n' \quad \langle n := v,\ M\rangle \to \langle P,\ M'\rangle}{\langle n := v,\ (n' \mapsto v') +\!\!+ M\rangle \to \langle P,\ (n' \mapsto v') +\!\!+ M'\rangle}$$

$$\frac{Holds([b,\ M]) \quad \langle P_0,\ M\rangle \to \langle P_0',\ M'\rangle}{\langle \texttt{if}\ (b)\ \texttt{then}\ P_0\ \texttt{else}\ P_1\ \texttt{fi},\ M\rangle \to \langle P_0',\ M'\rangle}$$

$$\frac{\neg Holds([b,\ M]) \quad \langle P_1,\ M\rangle \to \langle P_1',\ M'\rangle}{\langle \texttt{if}\ (b)\ \texttt{then}\ P_0\ \texttt{else}\ P_1\ \texttt{fi},\ M\rangle \to \langle P_1',\ M'\rangle}$$

$$\frac{Holds([b,\ M]) \quad \langle P_0,\ M\rangle \downarrow}{\langle \texttt{if}\ (b)\ \texttt{then}\ P_0\ \texttt{else}\ P_1\ \texttt{fi},\ M\rangle \downarrow} \qquad \frac{\neg Holds([b,\ M]) \quad \langle P_1,\ M\rangle \downarrow}{\langle \texttt{if}\ (b)\ \texttt{then}\ P_0\ \texttt{else}\ P_1\ \texttt{fi},\ M\rangle \downarrow}$$

$$\frac{\langle P_0,\ M\rangle \to \langle P_0',\ M'\rangle}{\langle P_0; P_1,\ M\rangle \to \langle P_0'; P_1,\ M'\rangle} \qquad \frac{\langle P_0,\ M\rangle \downarrow \quad \langle P_1,\ M\rangle \to \langle P_1',\ M'\rangle}{\langle P_0; P_1,\ M\rangle \to \langle P_1',\ M'\rangle}$$

$$\frac{\langle P_0,\ M\rangle \downarrow \quad \langle P_1,\ M\rangle \downarrow}{\langle P_0; P_1,\ M\rangle \downarrow} \qquad \frac{}{\langle \texttt{skip},\ M\rangle \downarrow}$$

The above semantics defines a transition relation $\to$ and a termination predicate $\downarrow$ on program states. Again the deduction rules should be read as: the transition in the conclusion can be made (or the predicate in the conclusion is valid) if the statements about transitions and/or predicates in the premises are valid.

## 2.2 Transition System Specification (TSS)

Transition System Specifications (TSS's), as presented by Groote and Vaandrager in [64], are formalizations of SOS. In this chapter, we define the concept of TSS in a more general setting including the concepts of multi-sorted signatures and terms as labels, mainly inspired by [48]. This general definition of TSS is the unifying framework for most of the material presented throughout this thesis. In each chapter, we define a simplified instance of this general framework and formulate our results around it.

**Definition 2.2 (Signatures, Terms and Substitutions)** Assume $\mathcal{S}$ to be a set of sorts. Fix a set of sorted *variables* $V = \{x, y, \ldots\}$ with infinitely many variables of each sort. The sets of variables of sort $S \in \mathcal{S}$ is denoted by $V_S$. A *signature* $\Sigma$ consists of pairs $(f, S_0 \times \ldots \times S_{n-1} \to S_n)$ (with $S_i \in \mathcal{S}$, for all $0 \leq i \leq n$) where the first component of the pair is called the *function symbol* and the second is its *arity*, denoted by $ar(f)$. We assume that for a function symbol $f$ there is at most one pair with the first component $f$ in $\Sigma$. Function symbols with an arity of the form $. \to S$ are called *constants*.

Henceforth, we write $\overrightarrow{X}_{n-1}$ for a list of size $n$ of elements, i.e., $X_0, \ldots, X_{n-1}$. We write $\overrightarrow{X}_n \in V$ and by that we mean $X_0 \in V \wedge \ldots \wedge X_n \in V$. We also write $\overrightarrow{X}_n R \overrightarrow{Y}_n$ and by that we mean $(X_i, Y_i) \in R$ for all $i$, $0 \leq i < n$. When we (syntactically) replace a list with another, we always assume that the substituted and substituting elements are of the same sorts.

*Terms* $t, t', t_0, \ldots \in \mathcal{T}(\Sigma, V)$ based on a signature $\Sigma$ and set of sorted variables $V$ is a set of sorted terms $\mathcal{T}_S(\Sigma, V)$ for all $S \in \mathcal{S}$ and is inductively defined as follows.

1. for all $x \in V_S$, $x \in \mathcal{T}_S(\Sigma, V)$;
2. for all $(f, ar(f)) \in \Sigma$, $ar(f) = S_0 \times \ldots \times S_{n-1} \to S \Rightarrow$ $\forall_{t_0 \in \mathcal{T}_{S_0}(\Sigma,V), \ldots, t_{n-1} \in \mathcal{T}_{S_{n-1}}(\Sigma,V)}\ f(\overrightarrow{t}_{n-1}) \in \mathcal{T}_S(\Sigma, V)$.

A *substitution* $\sigma : V \to \mathcal{T}(\Sigma, V)$ is a function replacing variables of a sort with terms of the same sorts. Substitutions are lifted to terms as expected.

The set of *closed terms* $p, q, p', p_0, \ldots \in \mathcal{C}(\Sigma)$ is the set of all terms that do not contain a variable. A substitution is *closed* if all terms in its range are closed terms.

We shall keep $\Sigma$ and $V$ fixed but arbitrary henceforth, so that we do not need to mention them. Hence, we write $\mathcal{T}$ and $\mathcal{C}$ and we mean $\mathcal{T}(\Sigma, V)$ and $\mathcal{C}(\Sigma)$, respectively, for fixed $\Sigma$ and $V$.

A transition system specification, defined below, is a logical way of defining a transition relation on (closed) terms. We need some important basic definitions first.

**Definition 2.3 (Transition System Specification (TSS))** A *Transition System Specification (TSS)* is a tuple $(\Sigma, V, Rel, Pr, D)$ of $\Sigma$ a signature, $Rel$ and $Pr$ disjoint sets of relations and predicates on terms with fixed arities, and $D$ a set of deduction rules.

For $r \in Rel$ of arity $n$, $t, t' \in \mathcal{T}$, and $\overrightarrow{t}_{n-1} \in \mathcal{T}$, call $t \overset{\overrightarrow{t}_{n-1}}{\rightarrow_r} t'$ a *positive* and $t \overset{\overrightarrow{t}_{n-1}}{\nrightarrow_r}$ a *negative transition formula*. We call $t$ the *source* of both transition formulae and $t'$ the *target* of the positive one.

For $P \in Pr$ of arity $n$, $t \in \mathcal{T}$, and $\overrightarrow{t}_{n-1} \in \mathcal{T}$, we call $P(\overrightarrow{t}_{n-1})\ t$ a *positive predicate formula* and $\neg P(\overrightarrow{t}_{n-1})\ t$ a *negative predicate formula*. A (positive or negative) *formula* is a (positive or negative) transition or predicate formula. We say formulae are *closed* when all the terms they mention are.

A *deduction rule* $dr \in D$ is a tuple $(H, c)$ where $H$ is a set of formulae and $c$ is a positive formula. We call $c$ the *conclusion* and formulae in $H$ *premises*. We write $(H, c)$ as $\frac{H}{c}$.

A TSS is called *positive* if it does not have a deduction rule with negative formulae among its premises.

Note that any transition relation of arity $n$ can be viewed as a predicate of arity $n+1$. [135] also shows how to code predicates in transition relations using formulae with dummy right-hand sides.

**Example 2.4** Consider the SOS specification of the simple programming language presented in Example 2.1. We can formalize this SOS specification by fixing sorts *Bool* for boolean expressions, *Prg* for program, *Mem* for memories, *BSt* for boolean expression states and *St* for program states. Then the signature of the TSS is defined as the following pairs of function symbols and arities.

| | |
|---|---|
| $(\texttt{skip}, \quad \rightarrow Prg)$ | $(n := v, \quad \rightarrow Prg)_{n \in Name, v \in Val}$ |
| $(\_;\_, \quad Prg \times Prg \rightarrow Prg)$ | $(\texttt{if}\ (\_)\ \texttt{then}\ \_\ \texttt{else}\ \_\ \texttt{fi}, \quad Bool \times Prg \times Prg \rightarrow Prg)$ |
| $(\top, \quad \rightarrow Bool)$ | $(n == v, \quad \rightarrow Bool)_{n \in Name, v \in Val}$ |
| $(\bot \quad \rightarrow Bool)$ | $(\_ \vee \_, \quad Bool \times Bool \rightarrow Bool)$ |
| $(\neg\_, \quad Bool \rightarrow Bool)$ | $((n \mapsto v) +\!\!+\ \_, \quad Mem \rightarrow Mem)_{n \in Name, v \in Val}$ |
| $(\texttt{nil}, \quad \rightarrow Mem)$ | $([\_, \_], \quad Bool \times Mem \rightarrow BSt)$ |
| | $(\langle \_, \_ \rangle, \quad Prg \times Mem \rightarrow St)$ |

In the above signature, *BSt* and *St* stand for operational states for boolean expressions and programs, respectively. The TSS has a transition relations $\rightarrow$ and two predicates *Holds* and $\downarrow$ all of arity zero. Deduction rules of the TSS are those given in Example 2.1.

## 2.3 The Semantics of a TSS

A TSS is supposed to induce a unique semantics, namely a unique set of positive (transition and predicate) formulae on closed terms. For positive TSS's, the set of induced positive formulae are precisely defined by those that have a proof using instances of deduction rules in the TSS. The following definition formalizes this concept.

**Definition 2.5 (Provable Positive Formulae)** A *proof* of a closed positive formula $\phi$ (in a positive TSS $tss$) is a well-founded upwardly branching tree of which the nodes are labelled by closed formulae such that

- the root node is labelled by $\phi$, and
- if $\psi$ is the label of a node $q$ and $\{\psi_i \mid i \in I\}$ is the set of labels of the nodes directly above $q$, then there exist a deduction rule $\frac{\{\chi_i \mid i \in I\}}{\chi}$ in $tss$ and a substitution $\sigma$ such that $\sigma(\chi) = \psi$, and for all $i \in I$, $\sigma(\chi_i) = \psi_i$.

A closed positive formula $\phi$ is provable in a $tss$, notation $tss \vdash \phi$, if there is a proof for it. The semantics of $tss$ is the minimal set containing all provable formulae. If $tss$ contains no predicates, its semantics is also referred to as *the transition relation(s) induced by* $tss$.

The introduction of negative premises poses an interesting and rather difficult question concerning the semantics of TSS's. In other words, it is not immediately clear what can be considered a "proof" for a negative formula.

The first generic answer to this question was formulated in [61, 25] which is the following notion of supported model.

**Definition 2.6 (Supported Model)** Consider a transition system specification $tss = (\Sigma, V, Rel, Pr, D)$ and a closed formula $\psi \in \mathcal{C}$; the *supported model* of the transition system specification is a set of closed positive formulae $\mathcal{M}$ satisfying the following constraint.

$$\psi \in \mathcal{M} \iff \left( \exists_{d \in D} d = \frac{H}{c} \wedge \exists_\sigma \forall_{h \in H} \mathcal{M} \vDash \sigma(h) \wedge \sigma(c) = \psi \right)$$

where $\mathcal{M} \vDash \phi$ depending on the form of $\phi$ has the following meanings:

- for positive formulae: $\mathcal{M} \vDash p \xrightarrow{\overrightarrow{p}_n}_r p'$ means that $p \xrightarrow{\overrightarrow{p}_n}_r p' \in \mathcal{M}$ and $\mathcal{M} \vDash P(\overrightarrow{p}_n)p$ means that $P(\overrightarrow{p}_n)p \in \mathcal{M}$;
- for negative formulae: $\mathcal{M} \vDash p \overset{\overrightarrow{p}_n}{\nrightarrow}_r$ means that there exists no $p' \in \mathcal{C}$ such that $p \xrightarrow{\overrightarrow{p}_n}_r p' \in \mathcal{M}$ and $\mathcal{M} \vDash \neg P(\overrightarrow{p}_n)p$ means that $P(\overrightarrow{p}_n)p \notin \mathcal{M}$.

The notion of supported model does not always coincide with the intuition. Two counter-intuitive supported models are illustrated in the following example.

**Example 2.7** Consider a signature with a sort $P$ for processes and constants $(a, \rightarrow P)$ and $(b, \rightarrow P)$.

$$\frac{a \xrightarrow{a} a}{a \xrightarrow{a} a} \quad \Bigg| \quad \frac{b \overset{b}{\nrightarrow}}{a \xrightarrow{a} a}, \quad \frac{a \overset{a}{\nrightarrow}}{b \xrightarrow{b} b}$$

Consider the above two TSS's with the signature given above and two sets of deduction rules given above (with a single transition relation $\rightarrow$ and no predicate). The TSS at the left-hand side induces two supported models namely, $\emptyset$ and $\{a \xrightarrow{a} a\}$. We believe that the empty set is the only justified model since there is no way to prove a transition for $a$.

The TSS at the right-hand side has two supported models $\{a \xrightarrow{a} a\}$ and $\{b \xrightarrow{b} b\}$. There is no good reason to choose among these two supported models and even both may be considered unjustified, since each of them reiles on a premise that has no good reason to hold.

Several alternatives to the notion of supported model have been proposed for which [59] provides an overview and a comparison. Here, we also quote the notion of *stable model* [27, 59], defined below, that gives a reasonable semantics for transition system specification with negative premises. As argued in [27], TSS's that do not have a unique stable model should be ruled out and considered pathological.

**Definition 2.8 (Stable Model)** A closed positive formula $\phi$ is *provable* from a set of positive formula $T$ and a transition system specification $tss$, denoted by $(T, tss) \vdash \phi$, if and only if there is an upwardly branching tree of which the nodes are labelled by closed formulae such that

- the root node is labelled by $\phi$,
- if the label of a node $q$, denoted by $\psi$, is a positive formula and $\{\psi_i \mid i \in I\}$ is the set of labels of the nodes directly above $q$, then there exist a deduction rule $\dfrac{\{\chi_i \mid i \in I\}}{\chi}$ in $tss$ (where $\chi_i$ can be a negative or a positive formula) and a substitution $\sigma$ such that $\sigma(\chi) = \psi$, and for all $i \in I$, $\sigma(\chi_i) = \psi_i$, and

- if the label of a node $q$, denoted by $\chi$, is a negative formula then $T \vDash \chi$ (as defined in Definition 2.6).

A *stable model* defined by a transition system specification $tss$ is a set of formulae $T$ such that for all closed positive formulae $\phi$, $\phi \in T$ if and only if $(T, tss) \vdash \phi$.

**Example 2.9** Consider the TSS's given in Example 2.7. The anomaly of TSS at the left-hand side is now resolved as the only stable model for this TSS is the empty set of formulae. The anomaly of the right-hand side TSS still remains for it admits two stable models $\{a \xrightarrow{a} a\}$ and $\{b \xrightarrow{b} b\}$. In the next section, we review a method called *stratification*, proposed by [27], that can guarantee a TSS to induce a unique stable model and thus, rule out pathological TSS's of this sort.

Notions of supported- and stable-model are extended to three-valued supported- and three-valued stable-model in the literature [27, 59] and SOS meta-theorems have been re-formulated in this more general setting. In [27], the notion of *positive after reduction* (also called *complete*, for example, in [59]) is defined as a criterium for well-defined-ness of the semantics and is shown to be more general than stratification.

We formulate most of our results based on the two-valued stable model semantics of TSS's and some of them based on the notion of stratification. However, we expect them to carry over to the more general settings of three-valued stable models and TSS's that are positive after reduction, respectively.

# Chapter 3

# Standard Formats for SOS

"The bottom line for mathematicians is that the architecture has to be right. In all the mathematics that I did, the essential point was to find the right architecture. It's like building a bridge. Once the main lines of the structure are right, then the details miraculously fit. The problem is the overall design."

[Freeman Dyson]

"The progress of Science consists in observing interconnections and in showing with a patient ingenuity that the events of this ever-shifting world are but examples of a few general relations, called laws. To see what is general in what is particular, and what is permanent in what is transitory, is the aim of scientific thought."

[Alfred North Whitehead]

---

An extended version of this chapter is to appear as: J.F. Groote, M.R. Mousavi, M.A. Reniers, A Hierarchy of SOS Rule Formats, In P. Mosses and I. Ulidowski eds., *Proceedings of the 2nd Workshop on Structural Operational Semantics (SOS'05)*, Lisbon, Portugal, Electronic Notes in Theoretical Computer Science, Elsevier, July 2005.

## 3.1 Introduction

By imposing syntactic restrictions on TSS's one can deduce several interesting properties about their induced operational semantics. These properties range from issues such as well-definedness of the operational semantics [61, 27, 59] to security- [120, 121] and probability-related issues [19, 73]. The syntactic restrictions imposed by these meta-theorems usually suggest particular forms of deduction rules to be safe for a particular purpose and hence these meta-theorems usually define what is called a *SOS standard format*. [5] provides an overview of existing SOS standard formats at its date of publication (2001). Since then, a number of new standard formats have been proposed and a fresh overview of the field can be beneficial. In this chapter we give an informal and partial overview of the field to date. When we feel the need, concerning what is to be presented in the rest of the thesis, we delve into details of a particular standard format and give a precise definition. Hence, our presentation of different standard formats may be a bit unbalanced.

The rest of this chapter is structured as follows. In Section 3.2, we present different syntactic features that TSS's in different frameworks may be allowed to have. These aspects will provide us with a natural classification of different SOS frameworks (classes of TSS's) defined in the literature. Then, in Section 3.3, we review several semantic meta-theorems formulated around these frameworks. Section 3.4 summarizes this chapter by presenting a lattice of existing standard formats, ordered by their syntactic features and annotated with their semantic meta-theorems.

## 3.2 Syntactic Features of TSS's

### 3.2.1 Labels

Labels are terms that may appear as parameters of transition relations and predicates in the deduction rules. SOS frameworks can be classified with respect to the kind of labels they afford as follows.

**Open Terms as Labels** Many SOS frameworks assume a special sort for labels and only allow for constants (alternatively, closed terms) of this sort to appear as labels. Such SOS frameworks thus forbid any correlation between valuation of terms and labels through the use of common variables. Frameworks defined in [48, 52, 21] and Definition 2.3 (used in Chapter 9 of this thesis) allow for arbitrary terms as labels. All other SOS frameworks reviewed in this chapter only allow for constant labels.

Open terms are used as labels in a number of cases in transition system specifications. For example, in Chapter 9, we treat the Calculus of Higher Order Communicating Systems (CHOCS) [119] which uses this feature.

**Lists of Terms as Labels** Most SOS frameworks only allow for a single term as label. The only existing exceptions are those of [48] and Chapter 9 of this thesis.

As noted above, TSS's with constant labels are by far the most common kind of TSS's in the literature and will often be used in the rest of this thesis. For notational convenience, in TSS's with constant labels, we separate the sort of states (called processes) and sort of labels in the definition of the TSS.

**Definition 3.1 (TSS's with Constant Labels)** A *TSS with constant labels* is a tuple $(\Sigma, V, L, \mathit{Rel}, \mathit{Pr}, D)$ where $\Sigma$ and $V$ are, as before, signatures and variables, $L$ is a set of labels, $\mathit{Rel}$ is a set of unary transition relations and $\mathit{Pr}$ is a set of unary predicates. For $t, t' \in \mathcal{T}(\Sigma, V)$, $l \in L$, $r \in \mathit{Rel}$ and $P \in \mathit{Pr}$, $t \xrightarrow{l}_r t'$ and $t \overset{l}{\nrightarrow}_r$ are *positive and negative transition formulae* with constant labels, respectively and $P(l)t$ and $\neg P(l)t$ are *positive and negative predicate* formulae with constant labels. Based on this restricted notion of formulae, deduction rules are defined in a similar way as in Definition 2.3.

The notion positive carries over to TSS's with constant labels naturally. If a TSS with constant labels has an empty set of predicates, we omit the $\mathit{Pr}$ part in the definition. Also, if a TSS has only one transition relation, we omit the $\mathit{Rel}$ part in the definition of TSS and the $r$ subscript in formulae, for brevity. Note that transitions and predicates without a label (e.g., those used in Example 2.1) can easily be coded in the above framework by taking a singleton set of labels with a dummy label as its only member.

### 3.2.2 Signatures

**Names and Binders** In many contemporary process algebras and calculi, concepts of names, (actual and formal) variables and name abstraction (binding) are present and even serve as a basic ingredient. For example, in the $\pi$-calculus of Milner, Parrow and Walker [89, 90, 91], names are first-class citizens and the whole calculus is built around the notion of passing names among concurrent agents. Less central, yet important, instances of these concepts appear in different process algebras in the form of the recursion operator, the infinite sum operator and the time-integration operator (cf., for example, [89], [79, 109] and [9], respectively). Hence, it is interesting to accommodate the concept of names in the TSS framework.

There have been a few attempts in this direction. In [48], an extension of the TSS framework of Definition 2.3 is given in which function symbols may name a list of binding variables in the definition of their arity. More precisely, arity of a function symbol has the form $\overrightarrow{S}_{i_0}.S_0 \times \ldots \times \overrightarrow{S}_{i_{n-1}}.S_{n-1} \rightarrow S_n$, where the list of sorts before each arguments are to be replaced by *actual variables* (names) that bind other instances of the same variables in the argument. Furthermore, the term structure is provided with an explicit substitution for replacing actual variables with (possibly open) terms.

Another proposal for modeling names and binders is formulated in [83, 84] which makes use of parameterized variables. Apart from the introduction of parameterized variables, the TSS framework of [83, 84] is more restricted than that of [48] and Definition 2.3 in that it does not allow for arbitrary terms as labels.

In the rest of this thesis, we do not treat the concept of names and binders. Also, apart from [48, 83, 84], all other standard formats we mention in the remainder of this chapter do not have this feature and hence, are restricted instances of Definition 2.3.

**Multi-Sorted States** Based on the number of sorts allowed in the signature, an SOS framework may be classified in the following three categories:

1. Multi-sorted TSS's: In such frameworks, there is no restriction on the sorts allowed for constructing terms.

2. $N$-sorted TSS's: A framework may only allow for a fixed number of sorts participating in the signature. An example of such frameworks appears in Chapter 8 of this thesis where there are two distinguished sorts of processes and data. Apart from these two sorts that are used to define the states of the semantics, there is a sort for constant labels.

3. Single-sorted: This is the most common framework in the literature. It has a single sort for operational states which is usually called the sort of *processes* (and terms from this sort are process terms). In this framework, there is usually a sort for constant labels, as well.

The TSS of [52] has a special status with respect to its allowed signatures. Namely, it requires a special sort for processes and at least one sort for labels. Furthermore, it requires that process sorts should not participate in function symbols with label sorts as targets.

### 3.2.3 Positive Premises and the Conclusion

**Look-Ahead** A framework allows for look-ahead if a deduction rule in the framework may have two premises with a variable in the target of one of the premises being present in the source of the other. An example of a deduction rule with look-ahead is the following.

$$\frac{x \xrightarrow{\tau} y \quad y \xrightarrow{l} z}{x \xrightarrow{l} z}$$

The above rule from [54] is used to combine silent ($\tau$) and ordinary transitions in order to implement a weak semantics (by ignoring silent steps) inside a strong semantic framework.

**Well-foundedness** Here, we formally define the concept of well-foundedness which is a useful concept in the remainder of this thesis.

**Definition 3.2 (Variable Dependency Graph and Well-foundedness)** The *variable dependency graph* of a deduction rule is a graph of which the nodes are variables and there is an edge between two variables if one appears in the source and the other in the target of the same positive premise in the deduction rule. A deduction rule is *well-founded* when all the backward chains of variables in the variable dependency graph are finite. A TSS is well-founded when all its deduction rules are.

All practical instances of SOS specifications are well-founded. Well-foundedness also comes very handy in the proof of semantic meta-results for SOS frameworks. Hence, it is only of theoretical interest whether a framework allows for non-well-founded deduction rules or not.

**Copying** A framework has the *copying* feature if it allows for repetition of variables in the target of the conclusion. A simple example of copying is the second rule in the following TSS which defines the semantics of the `while` construct (in the simple programming language framework developed in Example 2.4).

$$\frac{\neg \mathit{Holds}[b,\ M]}{\langle \texttt{while}\ (b)\ \texttt{do}\ P\ \texttt{od},\ M\rangle \downarrow}$$

$$\frac{\mathit{Holds}[b,\ M]}{\langle \texttt{while}\ (b)\ \texttt{do}\ P\ \texttt{od},\ M\rangle \rightarrow \langle P; \texttt{while}\ (b)\ \texttt{do}\ P\ \texttt{od},\ M\rangle}$$

**Infinite Premises** It is an interesting theoretical question whether a framework allows for an infinite number of premises or not. Also practically, when dealing with infinite domains (e.g., infinite basic actions, data or time domains), it is sometimes useful to have deduction rules with infinitely many premises. The following example from [100] illustrates a possible use of deduction rules with infinitely many premises:

$$\frac{x \xrightarrow{a} y \quad a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \qquad \frac{\forall_{a \in A \setminus H} x \stackrel{a}{\nrightarrow}}{\partial_H(x) \xrightarrow{\chi} \delta}$$

The above deduction rules define the semantics of the encapsulation operator $\partial_H(\_)$ which forbids its parameter from performing actions in $H$. If the parameter cannot perform any ordinary action allowed by $\partial_H$ then it makes a transition to the deadlocking process $\delta$. If the set of basic actions $A$ is infinite, then for each finite $H$, the deduction rule on the right-hand side has infinitely many (negative) premises.

### 3.2.4 Negative Premises

As illustrated in Chapter 2, negative premises are a complicating factor in SOS frameworks. We have already shown an example of the use of negative premises above. To our knowledge, the first example of negative premises in SOS appeared in [7] in the specification of the semantics of the following priority operator $\theta(\_)$.

$$\frac{x \xrightarrow{a} x' \quad \forall_{b > a} x \stackrel{b}{\nrightarrow}}{\theta(x) \xrightarrow{a} \theta(x')}$$

The above deduction rule states that a parameter of $\theta(\_)$ can perform a transition with label $a$ if no transition with a label $b$ of higher priority can be performed (according to a given ordering $>$). In addition to negative premises, the above deduction rule may have infinitely many premises if there are infinitely many basic actions that have priority over a given action $a$.

### 3.2.5 Predicates

Predicates are useful syntactic features which are used to specify phenomena such as termination or divergence. We have already shown an example of a termination predicate in Example 2.1.

### 3.2.6 Other Syntactic Features

**Ordering the Deduction Rules** One way to avoid the use of negative premises (and sometimes predicates) is by defining an order among deduction rules. Then, a deduction rule of a lower order may be applied to prove a formula only when there is no deduction rule with a higher order applicable. For example, the semantics of the priority operator defined in Section 3.2.4 can be expressed in terms of a number of rules of the following form

$$\frac{x \xrightarrow{a} x'}{\theta(x) \xrightarrow{a} \theta(x')}$$

with an ordering among such rules based on the ordering among labels. The semantics of the sequential composition operator can also be defined as follows.

$$\frac{x \xrightarrow{l} x'}{x; y \xrightarrow{l} x'; y} \qquad \frac{y \xrightarrow{l} y'}{x; y \xrightarrow{l} y'}$$

with the rule on the left-hand side being ordered above the right-hand side rule. This way, the second argument of sequential composition can take over, only when the first part cannot make a transition, i.e., has terminated (we do not consider unsuccessful termination or deadlock in this simple setting). The implications of introducing an order among deduction rules and its possible practical use are investigated in [103, 126]

**Equational Specifications** Structural congruences are equational addenda to SOS specifications which can define inherent properties of function symbols or define some function symbols in terms of the others. For example, the following equation specifies that the order of arguments in a parallel composition does not matter or in other words, that parallel composition is commutative.

$$x \mid\mid y \equiv y \mid\mid x$$

In Chapter 5 of this thesis, we study the addition of equational specifications to SOS specifications in detail.

## 3.3 Semantic Meta-Results

### 3.3.1 Congruence for Behavioral Equivalences

An SOS specification is supposed to define a transition system semantics for processes and programs. However, in most practical cases the induced transition systems contain details that are not observable by experiments and thus should not be considered relevant. A notion of *behavioral equivalence* thus defines the intended semantics of processes and programs by abstracting from these details and concentrating on the observable part of the behavior. Similarly, *behavioral pre-orders* define when particular system is a restricted implementation of the other.

There is a myriad of notions of behavioral equivalence and pre-order in the literature [57, 56]. It is very much desired for a notion of behavioral equivalence (pre-order) to be compositional or in technical terms to be a *congruence* (pre-congruence). Hence, a number of SOS rule formats have been developed that guarantee these notions to be a (pre-)congruence [64, 24, 23, 58]. In the remainder, we confine ourselves to single-sorted frameworks with constant labels. In such frameworks the arity of a function symbol can be conveniently expressed by a natural number (representing the number of parameters on the left-hand side of the arrow). The only congruence meta-theorems for multi-sorted frameworks are those of [52, 83, 48] and with open terms as labels are [52, 48] and Chapter 9 of this thesis.

We start by defining the notion of congruence.

**Definition 3.3 ((Pre-)Congruence)** An equivalence (pre-order) $R \subseteq \mathcal{T} \times \mathcal{T}$ is a *(pre-)congruence* with respect to a signature $\Sigma$ if and only if for all $(f, ar(f)) \in \Sigma$ and all $\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1} \in \mathcal{T}$, if $\overrightarrow{p}_{ar(f)-1}\ R\ \overrightarrow{q}_{ar(f)-1}$ then $f(\overrightarrow{p}_{ar(f)-1})\ R\ f(\overrightarrow{q}_{ar(f)-1})$.

The first congruence formats were defined for the notion of strong bisimilarity, defined below.

**Definition 3.4 (Bisimulation and Bisimilarity [102])** A relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is a *bisimulation* relation with respect to a set of transition relations *Rel* and a set of predicates $Pr$ if and only if $\forall_{p,q \in \mathcal{C}}\ pRq \Rightarrow \forall_{r \in Rel, P \in Pr, l \in L}$

1. $\forall_{p' \in \mathcal{C}}\ (p \xrightarrow{l}_r p' \Rightarrow \exists_{q' \in \mathcal{C}}\ q \xrightarrow{l}_r q' \wedge (p', q') \in R)$;
2. $\forall_{q' \in \mathcal{C}}\ (q \xrightarrow{l}_r q' \Rightarrow \exists_{p' \in \mathcal{C}}\ p \xrightarrow{l}_r p' \wedge (p', q') \in R)$;
3. $P(p) \Leftrightarrow P(q)$.

Two closed terms $p$ and $q$ are *bisimilar* if and only if there exists a bisimulation relation $R$ with respect to *Rel* and $Pr$ such that $(p, q) \in R$. Two closed terms $p$ and $q$ are *bisimilar* with respect to a transition system specification $tss$, denoted by $tss \vdash p \underline{\leftrightarrow} q$, if and only if they are bisimilar with respect to the semantics of $tss$.

There are good reasons for considering strong bisimilarity as an important notion of behavioral equivalence. Here, we mention a few.

1. Strong bisimilarity usually gives rise to elegant theories and it turns out that congruence formats for it are also much more elegant and compact than those for other (weaker) notions;

2. For finite state processes, strong bisimilarity can be checked very efficiently in practice [101] while some weaker notions are intractable [72];

3. Other notions can often be coded in terms of strong bisimilarity [54].

So, it is not surprising that the first standard congruence format was geared toward strong bisimilarity. This format was proposed by De Simone in [42]. The De Simone format uses the positive framework with constant labels and allows for deduction rules of the following form:

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} t} \quad [Pred(\overrightarrow{l_i}, l)].$$

where $x_i$ and $y_i$ are distinct variables ranging over process terms, $f$ is a function symbol from the signature (e.g., sequential composition, parallel composition, etc.), $I$ is a subset of the set $\{0, \ldots, ar(f) - 1\}$ (indices of arguments of $f$), $t$ is a process term that does not have repeated occurrences of any variable (so called *architectural* term, disallowing copying of variables), $l_i$'s and $l$ are constant labels and *Pred* is a predicate stating the relationship between the labels of the premises and the label of the conclusion. (It turns out that side conditions of this kind do not play any role in the congruence result and thus we do not mention them in the rest of this chapter.)

Bloom, Istrail and Meyer, in their study of the relationship between bisimilarity and completed-trace congruence [25], define an extension of the De Simone format, called *GSOS* (for Structural Operational Semantics with Guarded recursion), to capture *reasonable* language definitions. The GSOS format extends the De Simone format by allowing for copying and negative premises. The GSOS format, which will be used in Chapter 7, is formally defined as follows.

**Definition 3.5 (GSOS Format)** A deduction rule is in the *GSOS* format when it is of the following form:

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in I, 0 \leq j \leq m_i\} \cup \{x_j \overset{l_{jk}}{\nrightarrow} \mid j \in J, 0 \leq k \leq n_j\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} t}.$$

where $f$ is a function symbol, $x_i$ $(0 \leq i < ar(f))$ and $y_{ij}$'s $(i \in I$ and $j \leq m_i)$ are all distinct variables, $I$ and $J$ are subsets of $\{0, \ldots, ar(f) - 1\}$, $m_i$ and $n_j$ are natural numbers (to set an upper bound on the number of premises), $vars(t) \subseteq \{x_i, y_{jk} | i \in I \cup J, j \in I, k \leq m_i\}$ and $l_{ij}$'s, $l_{jk}$'s and $l$ are constant labels. A TSS is in the *GSOS* format when all its deduction rules are.

Another orthogonal extension of the De Simone format is called tyft/tyxt format and is first formulated in [64].[1] This format allows for look-ahead, copying and an infinite set of premises.

**Definition 3.6 (Tyft/tyxt Format [64])** A rule is in the tyft format if and only if it has the following form.

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_{r} t}$$

where $x_i$ and $y_i$ are all distinct variables (i.e., for all $i, i' \in I$ and $0 \leq j, j' < ar(f)$, $y_i \neq x_j$ and if $i \neq i'$ then $y_i \neq y_{i'}$ and if $j \neq j'$ then $x_j \neq x_{j'}$), $f$ is a function symbol from the signature, $I$ is a (possibly infinite) set of indices, $t$ and $t_i$'s are arbitrary terms and $l_i$'s and $l$ are constant labels.

A rule is in tyxt format if it is of the above form but the source of conclusion is a variable distinct from all targets of premises. A TSS is in the tyft format when all its deduction rules are. A TSS is in the tyft/tyxt format when all its deduction rules are either in the tyft or in the tyxt format.

Any TSS in the tyft/tyxt format can be reduced to an equivalent TSS (inducing the same transition relations) in the tyft format. We use the tyft format as our basis for Chapters 4-6. In [64], to prove congruence of strong bisimilarity for TSS's in the tyft/tyxt format, well-foundedness of the TSS is assumed. Later, in [45], it is shown that the well-foundedness constraint can be relaxed and that for every non-well-founded TSS in the tyft/tyxt format, a TSS exists that induces the same transition relation and is indeed well-founded. In most of our proofs in this thesis, we assume the well-foundedness of the transition system specifications. In most cases, we expect that our results will carry over to the non-well-founded setting.

**Theorem 3.7 (Congruence of Bisimilarity for Tyft/tyxt [64, 45])** For a TSS in tyft/tyxt format, strong bisimilarity is a congruence.

The merits of the two extensions were merged in [61] where negative premises were added to the tyft/tyxt format, resulting in the ntyft/ntyxt format.

**Definition 3.8 (Ntyft/ntyxt Format [61])** A rule is in the ntyft format if and only if it has the following form.

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\} \quad \{t_j \stackrel{l_j}{\nrightarrow}_{r_j} | j \in J\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_{r} t}$$

[1] Tyft/tyxt is a code representing the structure of symbols in the deduction rules, namely, a general term (t) in the source of the premises, a variable (y) in the target of the premises, a function symbol (f) or a variable (x) in the source of the conclusion and a term (t) in the target of the conclusion.

The same conditions as of the tyft format hold for the positive premises and the conclusion. There is no particular constraint on the terms appearing in the negative premises. Set $J$ is the (possibly infinite) set of indices of negative premises. An ntyft rule of the above form is called an *$f$-defining* rule. A rule is in the ntyxt format if it is of the above form but the source of conclusion is a variable distinct from all targets of premises. A TSS is in the ntyft format when all its deduction rules are. A TSS is in the ntyft/ntyxt format when all its deduction rules are either in the ntyft or in the ntyxt format.

As explained before, introduction of negative premises in the ntyft/ntyxt format brings about doubts regarding the well-definedness of the semantics. In the coming section, we give well-definedness criteria (from [61, 27]) for the semantics of TSS's in the ntyft/ntyxt format. Interestingly, these criteria turn out to be useful for proving congruence of bisimilarity, as well (see the following section). Well-foundedness assumption was also used in [61] and was shown to be redundant in [46].

Finally, the PATH format [12] for Predicates And Tyft/tyxt Hybrid format) and the PANTH format [135] (for Predicates And Negative Tyft/tyxt Hybrid format) extend tyft/tyxt and ntyft/ntyxt with predicates, respectively. A deduction rule in PANTH format may have predicates, negative predicates, transitions and negative transitions in its premises and a predicate or a transition in its conclusion.

In [83], the PANTH format is extended for multi-sorted variable binding. This covers the problem of operators such as recursion or choice over a time domain. The issue of binding operators for multi-sorted process terms is also briefly introduced in [5].

A number of other standard formats have been proposed for the congruence of weaker notions of bisimulation. A major example of such formats is the Cool languages format introduced in [23] which proves congruence of (rooted) weak and branching bisimulations. This format has recently been reformulated in [58] and extended to prove congruence of delay and $\eta$-bisimulations. In [55], the *ready simulation* format is proposed that induces congruence for ready simulation. This format is the ntyft/ntyxt format without the look-ahead feature. The ready simulation format is further restricted in [24] to obtain pre-congruence for readiness, ready traces and failures pre-order. Note that pre-congruence for a pre-order implies congruence for the corresponding equivalence (the kernel of the pre-order).

### 3.3.2 Well-definedness of the Semantics

In [61], Groote defines a criterion which guarantees a TSS in the ntyft/ntyxt format to induce a well-defined semantics. This criterion, defined below, is called (strict) stratification and is originally due to [53] in the setting of logic programming.

**Definition 3.9 (Stratification [61])** A stratification of a transition system specification *tss* is a function $\mathcal{S}$ from closed positive formulae to an ordinal such that for all deduction rules in *tss* of the following form:

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} t'_i | i \in I\} \quad \{t_j \overset{l_j}{\nrightarrow}_{r_j} | j \in J\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t}$$

and for all closed substitutions $\sigma$, $\forall_{i \in I} \mathcal{S}(\sigma(t_i \xrightarrow{l_i}_{r_i} t'_i)) \leq \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t))$ and $\forall_{j \in J}\ \forall_{t' \in \mathcal{T}}\ \mathcal{S}(\sigma(t_j \xrightarrow{l_j}_{r_j} t')) < \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t))$. A transition system specification is called *stratified* when there exists a stratification function for it. If the measure decreases also from the conclusion to the positive premises, then the stratification is called *strict*.

The following theorem shows useful properties of stratified TSS's.

**Theorem 3.10 (Stratification, Well-definedness and Congruence [61, 27])** The following statements hold.

1. A strictly stratified TSS in the ntyft/ntyxt format has a unique supported model.
2. A stratified TSS in the ntyft/ntyxt format has a unique stable model.
3. For a stratified TSS in the ntyft/ntyxt format bisimilarity is a congruence.

TSS's in the GSOS format are strictly stratified using a measure of size on terms in the source of the transition formulae. As a corollary of the above theorem, one may deduce that the semantics of a TSS in the GSOS format is well-defined and bisimilarity is a congruence for such a TSS. Of course, congruence of bisimilarity for the GSOS format had been directly proven in [25].

Definition 3.9 and Theorem 3.10 can easily be extended to the PANTH format (by interpreting predicates as transitions with dummy targets). Theorem 3.10 has been generalized (to the so-called, *positive after reduction* or *complete TSS's*) in a three-valued setting in [27].

### 3.3.3 Conservativity of Language Extensions

The operational semantics of languages may be extended by adding new pieces of syntax to the signature and new rules to the set of deduction rules. A number of meta-theorems have been proposed to check whether extensions do not change the behavior of the old language and whether they preserve equalities among old terms. Two general instances of such meta-theorems are formulated in [48, 84].

We review the results of [48] in this section, which gives the most detailed account of this issue. We simplify these results to single-sorted signatures without binding which is the framework that we use throughout this thesis.

To extend a language defined by a TSS, one may have to combine an existing signature with a new one. However, not all signatures can be combined into one as the arities of the function symbols may clash. To prevent this, we define two signatures to be *consistent* when they agree on the arity of the shared function symbols. In the remainder, we always assume that extended and extending TSS's are consistent. The following definition formalizes the concept of operational extension.

**Definition 3.11 (Extension of a TSS)** Consider TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$. The extension of $tss_0$ with $tss_1$, denoted by $tss_0 \cup tss_1$, is defined as $(\Sigma_0 \cup \Sigma_1, V, L_0 \cup L_1,\ D_0 \cup D_1)$.

Next, we define when an extension of a TSS is called operationally conservative.

**Definition 3.12 (Operational Conservativity [134])** Consider TSS's $tss_0 = (\Sigma_0,\ V, L_0,\ D_0)$ and $tss_1 = (\Sigma_1, V,\ L_1,\ D_1)$. If $\forall_{p \in \mathcal{C}(\Sigma_0, V)}\ \forall_{p' \in \mathcal{C}(\Sigma_0 \cup \Sigma_1, V)}\ \forall_{l \in L_0 \cup L_1}$ $tss_0 \cup tss_1 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_0 \vDash p \xrightarrow{l} p'$, then $tss_0 \cup tss_1$ is an *operationally conservative extension* of $tss_0$.

Next, we formulate sufficient conditions to prove operational conservativity. But before that, we need a few auxiliary definitions.

**Definition 3.13 (Source Dependency)** All variables appearing in the source of the conclusion of a deduction rule are called *source dependent*. A variable of a deduction rule is *source dependent* if it appears in a target of a premise of which all the variables of the source are source dependent. A premise is *source dependent* when all the variables appearing in it are source dependent. A rule is *source dependent* when all its variables are. A TSS is *source dependent* when all its rules are.

**Definition 3.14 (Reduced Rules)** For a deduction rule $d = (H, c)$, the reduced rule with respect to a signature $\Sigma$ is defined by $\rho(d, \Sigma) \doteq (H', c)$ where $H'$ is the set of all premises from $H$ which have a $\Sigma$-term as a source.

The following result, from [48], gives sufficient conditions for an extension of a TSS to be operationally conservative.

**Theorem 3.15 (Operational Conservativity Meta-Theorem [48])** Given two TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$, $tss_0 \cup tss_1$ is an operationally conservative extension of $tss_0$ if:

1. $tss_0$ is source dependent;

2. for all $d \in D_1$ at least one of the following holds:

   (a) the source of the conclusion has a function symbol in $\Sigma_1 \setminus \Sigma_0$, or

   (b) $\rho(d, \Sigma_0)$ has a source-dependent positive premise $t \xrightarrow{l} t'$ such that $l \notin L_0$ or $t' \notin \mathcal{T}(\Sigma_0, V)$.

In Chapter 6 of this thesis, we study meta-theorems of this kind in details. There, we formulate instances of conservativity meta-theorems which allow for extending the language behavior while keeping the behavioral equivalences on the old subset intact.

### 3.3.4 Generating Equational Theories

Equational theories are central notions to process algebras [13, 68, 87]. They capture the basic intuition behind the algebra, and the models of the algebra are expected to respect this intuition (e.g., the models induced by operational semantics). One of the added values of having equational theories is that they enable reasoning at the level of syntax without committing to particular models of the algebra. For example, when the behavioral model (e.g., the transition system semantics associated to a term) is infinite, these techniques may come very handy.

To establish a reasonable link between the operational model and the equational theory of the algebra, a notion of behavioral equality should be fixed. Ideally, the notion of behavioral equivalence should coincide with the closed derivations of the equational theory. One side of this coincidence is captured by the *soundness* theorem which states that all closed derivations of the equational theory are indeed valid with respect to the particular notion of behavioral equality. The other side of the coincidence, called *ground-completeness*, states that all induced behavioral equalities are derivable from the equational theory. These concepts are formalized below.

**Definition 3.16 (Equational Theory)** An *equational theory* or *axiomatization* $(\Sigma, V, E)$ is a set of equalities $E$ on a signature $\Sigma$ of the form $t = t'$, where $t, t' \in \mathcal{T}$. A closed instance $p = p'$, for some $p, p' \in \mathcal{C}$, is derivable from $E$, denoted by $E \vdash p = p'$ if and only if it is in the smallest congruence relation on closed terms induced by the equalities of $E$.

An equational theory $(\Sigma, V, E)$ is *sound* with respect to a TSS $tss$ (also on signature $\Sigma$) and a particular notion of behavioral equality $\sim$ if and only if for all $p, p' \in \mathcal{C}$, if $E \vdash p = p'$, then it holds that $tss \vdash p \sim p'$. It is *ground-complete* if the converse implication holds.

In [3, 2], an automatic method for generating sound and ground-complete equational theories from GSOS specifications is presented. This technique was extended in [16] to cater for explicit termination of processes. This approach, although more complicated in nature, gives rise to more intuitive and more compact sets of equations compared to the original approach of [3].

### 3.3.5 Other Meta-Results

**Non-Interference** Confidentiality is an important aspect of security and non-interference [60] is a well-studied means to guarantee end-to-end confidentiality. Non-interference means that a user with a lower confidentiality level cannot infer anything about the higher level information by interacting with the system (using lower-level methods that has in hand). In [120, 121] a standard format for non-interference is proposed which is based on the Cool languages format (in order to guarantee compositionality of non-intereference) and imposes further restrictions to assure that the lower-level behavior of the system does not change as a result of performing higher-level transitions.

**Decomposition of Logical Formulae** In [67], a logical framework, called *Hennessy-Milner logic* after the authors' names, is proposed. Hennessy-Milner logic can be used to reason about processes and characterize their equalities. In [74, 75] a meta-theory is developed that allows for decomposing Hennessy-Milner formulae using the structure of terms in a generic way by examining deduction rules of the process language in the De Simone format. This result has been improved in [47] and extended to the ready simulation format (ntyft/ntyxt format without look-ahead).

**Stochasticity** For probabilistic transition systems, it is essential to make sure that the sum of all probabilities belonging to the same distribution amounts to 1 (or zero). This is called (semi-)stochasticity. In [73], a restricted form of the De Simone format is proposed that guarantees semi-stochasticity. To avoid dealing with negative premises the format of [73] supports ordering on rules.

**Bounded Non-determinism** In [127], a standard format, by imposing restrictions on the De Simone format, is proposed which guarantees that the induced semantics affords only bounded non-determinism, i.e., each closed term has only a finite number of outgoing transitions. Fokkink and Duong Vu in [49] generalize the result of [127] to a far more general SOS framework.

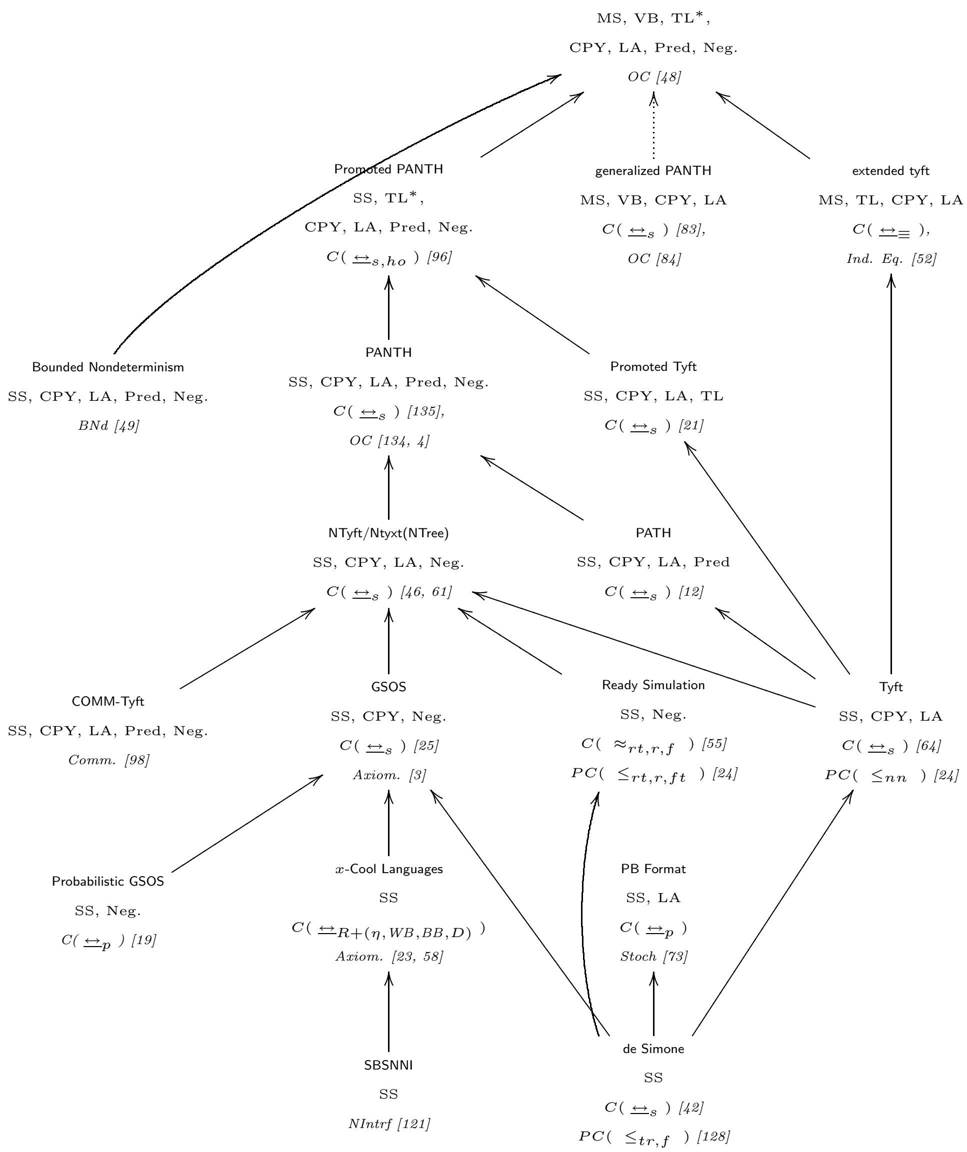


**Figure 3.1** An Overview of Existing SOS Frameworks

## 3.4 Summary and Conclusions

In this chapter, we provided an overview of SOS frameworks and existing meta-results about them. Figure 3.1 provides an overview of the frameworks and existing

Syntactic Features

| Shorthand | Syntactic Feature |
|---|---|
| SS vs. MS | Single- vs. Multi-Sorted Terms |
| VB | Variable Binding |
| TL vs. TL* | A Term vs. List of Terms as Labels |
| CPY | Copying Variables |
| LA | Look Ahead |
| Pred | Predicates |

Semantic Meta-Theorems

| Shorthand | Semantic Meta-Theorems |
|---|---|
| $C(\ \underline{\leftrightarrow}_x\ )$ | Congruence for $x$-Bisimulation |
| $C(\ \approx_x\ )$ | Congruence for $x$-equality |
| $PC(\ \leq_x\ )$ | Pre-congruence for $x$-pre-order |
| OC | Operational Conservativity |
| Axiom. | Deriving Sound and Complete Axiomatization |
| Ind. Eq. | Comparison of Induced Equality Classes |
| NIntrf | Non Intereference (Security-related [112]) |
| Bnd | Bounded Non-determinism |
| Stoch | Stochasticity |

**Table 3.1** Short-hands Used in Figure 3.1

results. The lattice presented there has SOS frameworks as nodes, ordered by syntactic inclusion (mainly based on the syntactic features). A node in this lattice has the following structure.

[Format Name]
Syntactic Features
*Semantic Meta-Theorems*

The short-hands used in this lattice are described in Table 3.1.

In the particular case of the generalized PANTH format, its relationship with the top element of the lattice is denoted by a dotted line; This is because a TSS in the generalized PANTH format does not syntactically fit in the framework of [48]. However, the syntactic features of this format are indeed subsumed by those of the framework of [48].

As it can be observed from Figure 3.1, many of the existing meta-results are formulated around restricted SOS frameworks and extending them to frameworks with more syntactic features are plausible topics for feature research. We conclude this chapter by mentioning a few interesting instances of such extensions:

- Extending the congruence rule formats for strong bisimilarity to a framework with both terms as labels and variable binding;
- Extending the congruence rule formats for weak bisimilarities to a setting with negative premises (e.g., ntyft format);
- Extending the axiomatization results to a setting with look-ahead.

# Chapter 4

# Commutativity

"Aus dem Leben
bin ich
in die Gedichte gegangen

Aus den Gedichten
bin ich
ins Leben gegangen

Welcher Weg
wird am Ende
besser gewesen sein?"

[Erich Fried]

An earlier version of this chapter has appeared as: M.R. Mousavi, M.A. Reniers, J.F. Groote, A Syntactic Commutativity Format for SOS, *Information Processing Letters (IPL)*, 93(5):217–223, Elsevier Science B.V., March 2005.

## 4.1 Introduction

Deriving algebraic axioms for SOS rules in [3, 2, 15, 125] are distinguished examples of SOS meta-theorems which generate a set of sound and (ground-)complete axioms for a given operational semantics in a syntactic format. Although commutativity axioms are derivable from the set of axioms generated by [3, 2, 15, 125], none of the approaches generate commutativity axioms explicitly and furthermore, they assume the existence of a number of standard constants and operators in the signature.

In this chapter, we aim at developing a meta-theorem for deriving commutativity axioms for certain operators in an SOS specification. Our format does not assume the presence of any special operator and builds upon a general congruence format, namely tyft [64]. The ultimate goal of this line of research is to develop the necessary theoretical background for a tool-set that can assist specifiers in developing Structural Operational Semantics for their languages, by proving different properties for the developed languages automatically.

The rest of this chapter is organized as follows. In Section 4.2, we start by presenting some preliminary notions about commutativity. Then, in Section 4.3, we give our proposal for a syntactic format for commutativity called comm-tyft (for commutative tyft). Section 4.4 addresses possible extensions of this format by adding tyxt rules, predicates and negative premises to the format (thus, achieving the expressivity of PANTH format [135]). Finally, Section 4.5 summarizes the results and presents concluding remarks. In this chapter, we use Definitions 2.2, 3.1, 3.3 and 3.4 without repeating them.

## 4.2 Basic Definitions

In this section, we define the notions of commutativity and the closure of commutativity under context. Here, we assume commutativity for all arguments of the operator. It is straightforward to restrict the results of this chapter to prove commutativity for a subset of arguments.

**Definition 4.1 (Commutativity)** A function symbol $f$ is called commutative on open (closed) terms with respect to a relation $R \subseteq \mathcal{T} \times \mathcal{T}$ if and only if for all $\overrightarrow{t}_{ar(f)-1} \in \mathcal{T}$ ($\overrightarrow{t}_{ar(f)-1} \in \mathcal{C}$) and for all $j$ and $k$ such that $0 \leq j < k < ar(f)$, $f(\overrightarrow{t}_{ar(f)-1}) \; R \; f(t_0, \ldots, t_k, \ldots, t_j, \ldots, t_{ar(f)-1})$. Function $f$ is called commutative (in our setting) if and only if it is commutative on closed terms with respect to bisimilarity.

To account for swapping of arguments of a commutative operator under arbitrary context, we define the notion of commutative congruence as follows.

**Definition 4.2 (Commutative Congruence)** Consider a set $COMM \subseteq \Sigma$. The commutative congruence relation $\sim_{cc}$ (w.r.t. $COMM$ and $\Sigma$) is the minimal relation satisfying the following requirements:

1. $\sim_{cc}$ is reflexive;
2. $\forall_{(f,ar(f))\in\Sigma}\ \forall_{\overrightarrow{p}_{ar(f)-1},\overrightarrow{q}_{ar(f)-1}\in\mathcal{T}}$
$\overrightarrow{p}_{ar(f)-1} \sim_{cc} \overrightarrow{q}_{ar(f)-1} \Rightarrow f(\overrightarrow{p}_{ar(f)-1}) \sim_{cc} f(\overrightarrow{q}_{ar(f)-1})$;
3. $\forall_{(g,ar(g))\in COMM}\ \forall_{\overrightarrow{p}_{ar(g)-1},\overrightarrow{q}_{ar(g)-1}\in\mathcal{T}}$
$\overrightarrow{p}_{ar(g)-1} \sim_{cc} \overrightarrow{q}_{ar(g)-1} \Rightarrow g(p_0,\ldots,p_k,\ldots,p_j,\ldots,p_{ar(g)-1}) \sim_{cc} g(\overrightarrow{q}_{ar(g)-1})$
(for all $0 \leq k < k < ar(g)$).

It can easily be seen that $\sim_{cc}$ is an equivalence relation and thus, it partitions the terms into equivalence classes $[t]_{cc}$. Two formulae $t_i \xrightarrow{l}_r t'_i$ and $t_j \xrightarrow{l}_r t'_j$ are called CC-equal if and only if $t_i \sim_{cc} t_j$ and $t'_i = t'_j$ (for technical reasons, the definition is asymmetric with respect to the source and target of the transition). By abusing the same notation, we denote the set of CC-equal formulae of $t_i \xrightarrow{l}_r t'_i$ by $[t_i \xrightarrow{l}_r t'_i]_{cc}$ and for a set $H$ of formulae we write $[H]_{cc}$ for $\bigcup_{h\in H}[h]_{cc}$.

As it appears from its name, there is more to Definition 4.2 than only commutativity. It also exploits congruence to switch arguments of operators in $COMM$ in an arbitrary context. The reason for adding congruence to commutativity is to make our commutativity format as general as possible. By taking the minimal reflexive and commutative relation (on open terms) instead of $\sim_{cc}$, one can obtain a simpler commutativity format compared to what we define in the remainder (and a simpler proof for it). This simpler format works equally well for the examples that we have checked so far. However, we prefer the existing formulation for its generality in order to cope with more possible applications in the future. The following example illustrates Definition 4.2.

**Example 4.3** Suppose that $\Sigma$ contains a binary operator $||$ and $|| \in COMM$; according to Definition 4.2, for variables $x_0, y_0, x_1, x_2$, we have $[x_0 \mid\mid (x_1 \mid\mid x_2)]_{cc} = \{x_0 \mid\mid (x_1 \mid\mid x_2),\ x_0 \mid\mid (x_2 \mid\mid x_1),\ (x_1 \mid\mid x_2) \mid\mid x_0,\ (x_2 \mid\mid x_1) \mid\mid x_0\}$. Furthermore, we have $(x_2 \mid\mid x_1) \mid\mid x_0 \xrightarrow{l}_r y_0 \in [x_0 \mid\mid (x_1 \mid\mid x_2) \xrightarrow{l}_r y_0]_{cc}$.

The following lemma shows that $\sim_{cc}$ is preserved under substitutions that respect $\sim_{cc}$.

**Lemma 4.4** For all $t, t' \in \mathcal{T}$ and for all $\sigma, \sigma' : V \to \mathcal{T}$, if $t \sim_{cc} t'$ (with respect to $COMM$), and for all $x \in vars(t)$, $\sigma(x) \sim_{cc} \sigma'(x)$, then $\sigma(t) \sim_{cc} \sigma'(t')$.

*Proof.* By an induction on the structure of $\sim_{cc}$:

- If the pair $(t, t')$ is in $\sim_{cc}$ due to reflexivity, then the lemma can be proven by a straightforward induction on the size of $t$. If $t$ and $t'$ are both a constant or both a variable, then the lemma holds trivially; otherwise, if $t = t' = f(\overrightarrow{t}_{ar(f)-1})$, then it follows from the induction hypothesis that $\sigma(t_i) \sim_{cc} \sigma'(t_i)$ and since relation $\sim_{cc}$ is closed under congruence, we have: $f(\sigma(\overrightarrow{t}_{ar(f)-1})) \sim_{cc} f(\sigma'(\overrightarrow{t}_{ar(f)-1}))$ and hence $\sigma(t) \sim_{cc} \sigma'(t)$.
- If $t = f(\overrightarrow{t}_{ar(f)-1})$ and $t' = f(\overrightarrow{t'}_{ar(f)-1})$, where $\overrightarrow{t}_{ar(f)-1} \sim_{cc} \overrightarrow{t'}_{ar(f)-1}$, then according to the induction hypothesis, we have $\sigma(t_i) \sim_{cc} \sigma'(t'_i)$ and it follows from the same constraint that $f(\sigma(\overrightarrow{t}_{ar(f)-1})) \sim_{cc} f(\sigma'(\overrightarrow{t'}_{ar(f)-1}))$. Hence, $\sigma(t) = f(\sigma(\overrightarrow{t}_{ar(f)-1})) \sim_{cc} f(\sigma'(\overrightarrow{t'}_{ar(f)-1})) = \sigma'(t')$.
- If $t = f(t_0, \ldots, t_k, \ldots, t_j, \ldots, t_{ar(f)-1})$ and $t' = f(\overrightarrow{t}_{ar(f)-1})$ (for arbitrary $0 \leq j < k < ar(f)$) , where and $\overrightarrow{t}_{ar(f)-1} \sim_{cc} \overrightarrow{t'}_{ar(f)-1}$ then according to the induction hypothesis, $\sigma(\overrightarrow{t}_{ar(f)-1}) \sim_{cc} \sigma'(\overrightarrow{t'}_{ar(f)-1})$ and by the same constraint $f(\sigma(t_0), \ldots, \sigma(t_k), \ldots, \sigma(t_j), \ldots, \sigma(t_{ar(f)-1})) \sim_{cc} f(\sigma'(\overrightarrow{t}_{ar(f)-1})$. Hence, $\sigma(t) \sim_{cc} \sigma'(t')$.

$\boxtimes$

## 4.3 Standard Format for Commutativity

We set our starting point from a standard congruence format (namely, tyft format) because in parts of our proofs, congruence is an essential ingredient.

**Definition 4.5 (Comm-tyft)** A transition system specification is in the comm-tyft format with respect to a set of function symbols $COMM \subseteq \Sigma$ if all its deduction rules are in tyft format and for every $f$-defining rule with $(f, ar(f)) \in COMM$ of the following form

$$\textbf{(d)}\ \frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_{r} t}$$

for which we denote the set of premises of **(d)** by $H$ and the conclusion by $c$, and for all $0 \leq j < k < ar(f)$, there exists a deduction rule **(d')** of the following form in the transition system specification

$$\textbf{(d')}\ \frac{H'}{f(\overrightarrow{x'}_{ar(f)-1}) \xrightarrow{l}_{r} t'}$$

and a bijective mapping (substitution) $\hbar$ on variables such that

- $\hbar(x'_i) = x_i$ for $0 \leq i < ar(f)$, $i \neq j$ and $i \neq k$,
- $\hbar(x'_j) = x_k$ and $\hbar(x'_k) = x_j$,
- $\hbar(t') \sim_{cc} t$,
- $\forall_{h' \in H'}\ \hbar(h') \in [H]_{cc} \cup \{c\}$.

Deduction rule **(d')** is called the *commutative mirror* of **(d)** (on arguments $j$ and $k$).

To put it informally, the role of substitution $\hbar$ in this definition is to account for the single swapping in the source of the conclusion and a possible isomorphic renaming of variables. Thus, the above format requires that for each $f$-defining rule, there exists a commutative mirror which firstly, has the same source of the conclusion as the original deduction rule with one swapping in the arguments, secondly, has the same source of the premises and target of the conclusion as the original rule up to arbitrary swapping of the arguments of commutative function symbols, and finally, may have the conclusion of the original rule as one of its premises. Next, we state that the comm-tyft format indeed induces commutativity for the set of operators under consideration.

**Theorem 4.6 (Commutativity for comm-tyft)** If a transition system specification is in the comm-tyft format with respect to a set of operators $COMM$, then all operators in $COMM$ are commutative.

*Proof.* Let $R$ be the relation $\sim_{cc}$ restricted to closed terms. By definition, this relation contains all the desired pairs of terms of the form $(f(\overrightarrow{p}_{ar(f)-1}), f(p_0, \ldots, p_k, \ldots, p_j, \ldots, p_{ar(f)-1}))$, where $0 \leq j < k < ar(f)$. Then, it only remains to prove that $R$ is a bisimulation relation.

Consider an arbitrary pair $(p, q) \in R$. Suppose that $p \xrightarrow{l}_r p'$ for some $r$, $l$, $p'$. We have to prove the existence of a $q'$ such that $q \xrightarrow{l}_r q'$ and $(p', q') \in R$ (and vice versa, the proof of which we omit due to symmetry). We start with a case distinction using the structure of $R$ in Definition 4.2.

1. For the reflexivity part of $R$, the theorem holds trivially.
2. For the congruence part, the theorem follows due to a similar construction as that of [64] since comm-tyft is a restriction of the tyft format. We quote the following Lemma from [64] which is a necessary ingredient of our proof.

   **Lemma 4.7** Consider a relation $R \subseteq \mathcal{T} \times \mathcal{T}$ which is closed under congruence. If for substitutions $\sigma$ and $\sigma'$ and a term $t \in \mathcal{T}$, it holds that $\forall_{x \in vars(t)} \sigma(x)\ R\ \sigma'(x)$, then $\sigma(t)\ R\ \sigma'(t)$.

Since $(p, q)$ is in $R$ due to congruence, we have that $p = f(\overrightarrow{p}_{ar(f)-1})$, $q = f(\overrightarrow{q}_{ar(f)-1})$, for some $(f, ar(f)) \in \Sigma$ such that $\overrightarrow{p}_{ar(f)-1}\ R\ \overrightarrow{q}_{ar(f)-1}$. We proceed with an induction on the depth of the proof for transition $p \xrightarrow{l}_r p'$.

If the transition has a proof of depth one, then there is a rule **(d)** of the following form:

$$\textbf{(d)}\frac{}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t}$$

and a substitution $\sigma$ such that $\sigma(x_i) = p_i$ $(0 \leq i < ar(f))$ and $\sigma(t) = p'$. By defining a new substitution $\sigma'$ as:

$$\sigma'(x) \doteq \begin{cases} q_i & \text{if } x = x_i (0 \leq i < ar(f)) \\ \sigma(x) & \text{otherwise} \end{cases}$$

we have a proof for $f(\overrightarrow{q}_{ar(f)-1}) \xrightarrow{l}_r \sigma'(t)$ using **(d)** and $\sigma'$. It follows from the definition of $\sigma'$ that $\forall_{x \in V}\ \sigma(x)\ R\ \sigma'(x)$ and thus according to Lemma 4.7, $\sigma(t)\ R\ \sigma'(t)$ and this concludes the induction basis.

For the induction hypothesis, suppose that transition $p \xrightarrow{l}_r p'$ has a proof of depth $n$ due to the following rule (as the root of its proof tree):

$$\textbf{(d)}\frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t}$$

and a substitution $\sigma$ such that $\sigma(x_i) = p_i$ $(0 \leq i < ar(f))$ and $\sigma(t) = p'$. Let $X \doteq \{x_i | 0 \leq i < ar(f)\}$ and $Y \doteq \{y_i | i \in I\}$. Our aim is to define a new substitution $\sigma'$ in such a way that $\forall_{x \in V}\ \sigma(x)\ R\ \sigma'(x)$. We start with the following partial definition:

$$\sigma'(x) \doteq \begin{cases} q_i & \text{if } x = x_i \quad (0 \leq i < ar(f)) \\ \sigma(x) & \text{if } x \in V \setminus (X \cup Y) \end{cases}$$

Hitherto, we already have that $\forall_{x \in V \setminus Y}\ \sigma(x)\ R\ \sigma'(x)$. It remains to complete the definition of $\sigma'$ for variables in $Y$ in an appropriate way. Take a premise of which $\sigma'$ is defined on all variables in the source (such a premise should exist due to our well-foundedness assumption about premises, see Definition 3.6). Suppose that one such a premise is $t_i \xrightarrow{l_i}_{r_i} y_i$, for some $i \in I$. For all $x \in vars(t_i)$, $\sigma'$ is already defined in appropriate way, thus $\sigma(x)\ R\ \sigma'(x)$. It follows from Lemma 4.4 that $\sigma(t_i)\ R\ \sigma'(t_i)$. Transition $\sigma(t_i) \xrightarrow{l_i}_{r_i} \sigma(y_i)$ has a proof of depth $n - 1$ or less and since $\sigma(t_i)\ R\ \sigma'(t_i)$, it follows from the induction hypothesis that there exists a $p'_i$ such that $\sigma'(t_i) \xrightarrow{l}_r p'_i$ and $\sigma(y_i)$ $R\ p'_i$. We define $\sigma'(y_i) \doteq p'_i$ and hence it holds that $\sigma(y_i)\ R\ \sigma'(y_i)$. This way, we complete the proof for $\sigma'(t_i \xrightarrow{l_i}_{r_i} y_i)$. Defining $\sigma'$ for each $y_i$ inductively, in this manner, concludes the proof since rule **(d)** together with $\sigma'$ gives us a proof for $q \xrightarrow{l}_r \sigma'(t)$ and according to the construction of $\sigma'$ (which *respects R on variables*, i.e., preserves the property $\forall_{x \in V}\ \sigma(x)\ R\ \sigma'(x)$), it follows from Lemma 4.7 that $\sigma(t)\ R\ \sigma'(t)$.

3. It remains to prove the theorem for the case where $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(q_0, \ldots,\ q_k, \ldots, q_j, \ldots, q_{ar(f)-1})$ for $(f, ar(f)) \in COMM$ and some $0 \leq j < k < ar(f)$ such that $\overrightarrow{p}_{ar(f)-1}\ R\ \overrightarrow{q}_{ar(f)-1}$.

We prove this by an induction on the depth of the proof for $p \xrightarrow{l}_r p'$. For the induction basis, suppose that $p$ can make a transition with a proof of depth 1. Then, this transition is due to an $f$-defining rule of the following form:

$$\textbf{(d)}\ \frac{}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t}$$

and a substitution $\sigma$ such that $\sigma(x_i) = p_i$ and $\sigma(t) = p'$. According to Definition 4.5, another rule exists in the transition system specification of the following form:

$$\textbf{(d')}\ \frac{H}{f(\overrightarrow{x'}_{ar(f)-1}) \xrightarrow{l}_r t'}$$

and a bijective mapping $\hbar$ such that

- $\hbar(x'_i) = x_i$ for $0 \leq i < ar(f)$, $i \neq j$ and $i \neq k$
- $\hbar(x'_j) = x_k$ and $\hbar(x'_k) = x_j$
- $\hbar(t') \sim_{cc} t$
- $\forall_{h \in H}\ \hbar(h) \in \{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t\}$

We define a new substitution $\sigma'$ as follows:

$$\sigma'(x) \doteq \begin{cases} q_i & \text{if } x = x'_i \land 0 \leq i < ar(f) \land i \neq j \land i \neq k \\ q_j & \text{if } x = x'_k \\ q_k & \text{if } x = x'_j \\ (\sigma \circ \hbar)(x) & \text{otherwise} \end{cases}$$

It immediately follows from the above definition that for all $x \in V$, $(\sigma \circ \hbar)(x)\ R\ \sigma'(x)$ (where $\circ$ denotes composition of mappings). It also follows from the definition of $\sigma'$ that $\sigma'(f(\overrightarrow{x'}_{ar(f)-1})) = q$. The last constraint of the comm-tyft format implies that $H$ is such that $\forall_{h \in H} \hbar(h) \in \{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} t\}$. Otherwise said, $H$ is either the empty set or $H = \{\hbar^{-1}(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t)\}$. If $H$ is empty, then we already have a proof for $q \xrightarrow{l}_r \sigma'(t')$ using deduction rule **(d')** and substitution $\sigma'$. Also, if $H =$

$\{\hbar^{-1}(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t)\}$ the proof for $q \xrightarrow{l}_r \sigma'(t')$ is complete since we already have a proof for $\sigma'(\hbar^{-1}(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t))$, using rule **(d)** and substitution $\sigma' \circ \hbar^{-1}$. Since $\hbar(t') \sim_{cc} t$ (thus, $\hbar^{-1}(t) \sim_{cc} t'$), and for all $x \in V$, $(\sigma \circ \hbar)(x) \sim_{cc} \sigma'(x)$, it follows from Lemma 4.4 that $\sigma(t) \sim_{cc} ((\sigma' \circ \hbar) \circ \hbar^{-1})(t')$. Hence, we have $\sigma(t)\ R\ \sigma'(t')$ (both $\sigma(t)$ and $\sigma(t')$ are closed terms and $R$ is $\sim_{cc}$ restricted to closed terms). This concludes the induction basis.

For the induction step, suppose that the theorem holds for all transitions with proofs of depth less than $n$. Then, consider a transition $p \xrightarrow{l}_r p'$ with a proof of depth $n$. This transition must be due to an $f$-defining rule **(d)** of the following form:

$$\textbf{(d)}\frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t}$$

and a substitution $\sigma$ such that $\sigma(x_i) = p_i$ and $\sigma(t) = p'$. We refer to the set of premises of **(d)** as $H$ and its conclusion as $c$. According to the Definition 4.5, there exists another rule **(d')** in the transition system specification of the following form:

$$\textbf{(d')}\frac{\{t'_j \xrightarrow{l'_j}_{r'_j} y'_j | j \in J\}}{f(\overrightarrow{x'}_{ar(f)-1}) \xrightarrow{l}_r t'}$$

and a mapping $\hbar$ of variables such that

- $\hbar(x'_i) = x_i$ for $0 \leq i < ar(f)$, $i \neq j$ and $i \neq k$
- $\hbar(x'_j) = x_k$ and $\hbar(x'_k) = x_j$
- $\hbar(t') \sim_{cc} t$
- $\forall_{j \in J}\ \hbar(t'_j \xrightarrow{l'_j}_{r'_j} y'_j) \in [H]_{cc} \cup \{c\}$

Let $X' = \{x'_i | 0 \leq i < ar(f)\}$ and $Y' = \{y'_j | j \in J\}$ be the set of variables in the source of the conclusion and the targets of the premises of deduction rule **(d')**, respectively. We aim at defining a new substitution $\sigma'$ such that for all variables $x \in V$, $(\sigma \circ \hbar)(x)\ R\ \sigma'(x)$. Similar to the induction basis, we start with the following (partial) definition for $\sigma'$:

$$\sigma'(x) \doteq \begin{cases} q_i & \text{if } x = x'_i \wedge 0 \leq i < ar(f) \wedge i \neq j \wedge i \neq k \\ q_j & \text{if } x = x'_k \\ q_k & \text{if } x = x'_j \\ (\sigma \circ \hbar)(x) & \text{if } x \in V \setminus (X' \cup Y') \end{cases}$$

To this end, we have $\sigma'(f(\overrightarrow{x'}_{ar(f)-1})) = q$ and $\sigma'$ is defined on all variables, except for those in $Y'$ and it satisfies $\forall_{x \in V \setminus Y'}\ (\sigma \circ \hbar)(x)\ R\ \sigma'(x)$.

For the variables in $Y'$, we complete the definition of $\sigma'$ by taking a premise whose source variables are all defined under $\sigma'$ and defining its target (such a premise should exist due to our well-foundedness assumption about premises, see Definition 3.6). Suppose that one of such premises is $t'_j \overset{l'_j}{\rightarrow}_{r'_j} y'_j$, for some $j \in J$. It follows from the last requirement on $\hbar$ in Definition 4.2 that either $\hbar(t'_j \overset{l'_j}{\rightarrow}_{r'_j} y'_j) = f(\overrightarrow{x}_{ar(f)-1}) \overset{l}{\rightarrow}_r t$ meaning that $t'_j = \hbar^{-1}(f(\overrightarrow{x}_{ar(f)-1}))$, $y'_j = \hbar^{-1}(t)$, or there exists an $i \in I$ such that $\hbar(t'_j \overset{l'_j}{\rightarrow}_{r'_j} y'_j) = t_i \overset{l_i}{\rightarrow}_{r_i} y_i$, which means that $t'_j \sim_{cc} \hbar^{-1}(t_i)$ and $y'_j = \hbar^{-1}(y_i)$.

In the former case, we have $\sigma'(t'_j) = (\sigma' \circ \hbar^{-1})(f(\overrightarrow{x}_{ar(f)-1})) = \sigma'(f(x'_0, \ldots, x'_j, \ldots, x'_k, \ldots, x'_{ar(f)-1})) = f(\overrightarrow{q}_{ar(f)-1})$. Since we know that $\overrightarrow{p}_{ar(f)-1}\ R\ \overrightarrow{q}_{ar(f)-1}$, and $f(\overrightarrow{p}_{ar(f)-1}) \overset{l}{\rightarrow}_r p'$ it follows from the congruence part of this proof that there exists a $q'$ such that $\sigma'(t'_j) \overset{l}{\rightarrow}_r q'$ and $p'\ R\ q'$. Then, we define $\sigma'(y'_j) \doteq q'$ and we fulfill the requirement that $(\sigma \circ \hbar)(y'_j)\ R\ \sigma'(y'_j)$ (since $\sigma(y_i) = p'$ and $y_i = \hbar(y'_j)$).

In the latter case, since $\hbar^{-1}(t_i) = t'_j$ (thus $\hbar^{-1}(t_i) \sim_{cc} t'_j$) and for all $x \in vars(t'_j)$, $(\sigma \circ \hbar)(x) \sim_{cc} \sigma'(x)$, it follows from Lemma 4.4 that $(\sigma \circ \hbar)(\hbar^{-1}(t_i)) \sim_{cc} \sigma'(t'_j)$ and thus $\sigma(t_i)\ R\ \sigma'(t'_j)$. Transition $\sigma(t_i) \overset{l_i}{\rightarrow}_{r_i} \sigma(y_i)$ has a proof of depth $n-1$ or less and since $\sigma(t_i)\ R\ \sigma'(t'_j)$, it follows from the induction hypothesis that there exists a $p'_j$ such that $\sigma'(t'_j) \overset{l}{\rightarrow}_r p'_j$ and $\sigma(y_i)\ R\ p'_j$. We define $\sigma'(y'_j) \doteq p'_j$ and hence it holds that $((\sigma \circ \hbar)(y'_j)\ R\ \sigma'(y'_j)$. This way, we complete the proof for $\sigma'(t'_j \overset{l}{\rightarrow}_r y'_j)$. Defining $\sigma'$ for each $y'_j$ inductively, in this manner, concludes the proof since rule **(d')** together with $\sigma'$ gives us a proof for $q \overset{l}{\rightarrow}_r \sigma'(t')$ and according to the construction of $\sigma'$ (which preserves the property $\forall_{x \in V}\ (\sigma \circ \hbar)(x) \sim_{cc} \sigma'\ (x))$ and since $\hbar^{-1}(t) \sim_{cc} t'$, it follows from Lemma 4.4 that $(\sigma \circ \hbar)(\hbar^{-1}(t)) \sim_{cc} (\sigma \circ \hbar)(t')$ and hence $\sigma(t)\ R\ \sigma'(t')$.

$\boxtimes$

In the remainder, we illustrate the idea behind our format by a few examples from the area of process algebra.

**Example 4.8 (Parallel Composition: Standard Rules)** Consider the following transition system specification for a parallel composition operator [13]:

$$\textbf{(p0)}\,\frac{x_0 \xrightarrow{l}_{r_0} y_0}{x_0 \mid\mid x_1 \xrightarrow{l}_{r_0} y_0 \mid\mid x_1} \qquad \textbf{(p1)}\,\frac{x_1 \xrightarrow{l}_{r_0} y_1}{x_0 \mid\mid x_1 \xrightarrow{l}_{r_0} x_0 \mid\mid y_1}$$

$$\textbf{(p2)}\,\frac{x_0 \xrightarrow{l_0}_{r_0} y_0 \quad x_1 \xrightarrow{l_1}_{r_0} y_1}{x_0 \mid\mid x_1 \xrightarrow{l}_{r_0} y_0 \mid\mid y_1} \quad comm(l_0, l_1) = l$$

If the synchronization function *comm* is commutative, then the above TSS is in the comm-tyft format w.r.t. the singleton set $COMM = \{\mid\mid\}$. This can be seen as follows. All deduction rules are obviously in tyft format. Deduction rule **(p1)** is the commutative mirror of **(p0)** (and vice versa) by using the mapping $\hbar$ defined by $\hbar(x_0) = x_1$, $\hbar(x_1) = x_0$, and $\hbar(y_1) = y_0$ and for the sake of symmetry $\hbar(y_0) = y_1$. Observe that $\hbar(x_0 \mid\mid y_1) = x_1 \mid\mid y_0 \sim_{cc} y_0 \mid\mid x_1$ and $\hbar(x_1 \xrightarrow{l}_{r_0} y_1) = x_0 \xrightarrow{l}_{r_0} y_0 \sim_{cc} x_0 \xrightarrow{l}_{r_0} y_0$. Similarly, deduction rule **(p0)** is the commutative mirror of **(p1)**, using the inverse of $\hbar$ (which is $\hbar$ itself) as a variable mapping. Finally, deduction rule **(p2)** is the commutative mirror of itself by using the mapping $\hbar$ such that $\hbar(x_0) = x_1$, $\hbar(x_1) = x_0$, $\hbar(y_0) = y_1$ and $\hbar(y_1) = y_0$. It is not obvious that this mapping satisfies the requirements due to the fact that the premises have different labels. In this case this is not a problem as the function *comm* is commutative. In fact, the instance of this deduction rule where $l_0$ and $l_1$ are instantiated by labels $a$ and $b$ matches with an instance where these are instantiated by $b$ and $a$ respectively.

**Example 4.9 (Parallel Composition: Non-Standard Rules)** For the reason of finite axiomatization, parallel composition may be defined in terms of auxiliary operators, namely the left merge $\mathbin{\|\!\_}$ and the communication merge $\mid$. Although the left merge is not a commutative operator, comm-tyft is still applicable for the following transition system specification of parallel composition, if one takes $COMM = \{\mid\mid, \mid\}$.

$$\textbf{(p0)}\,\frac{x_0 \mathbin{\|\!\_}\, x_1 \xrightarrow{l}_{r} y_0}{x_0 \mid\mid x_1 \xrightarrow{l}_{r} y_0} \qquad \textbf{(p1)}\,\frac{x_1 \mathbin{\|\!\_}\, x_0 \xrightarrow{l}_{r} y_0}{x_0 \mid\mid x_1 \xrightarrow{l}_{r} y_0} \qquad \textbf{(p2)}\,\frac{x_0 \mid x_1 \xrightarrow{l}_{r} y_0}{x_0 \mid\mid x_1 \xrightarrow{l}_{r} y_0}$$

$$\textbf{(lp0)}\,\frac{x_0 \xrightarrow{l}_{r} y_0}{x_0 \mathbin{\|\!\_}\, x_1 \xrightarrow{l}_{r} y_0 \mid\mid x_1} \qquad \textbf{(cp0)}\,\frac{x_0 \xrightarrow{l_0}_{r} y_0 \quad x_1 \xrightarrow{l_1}_{r} y_1}{x_0 \mid x_1 \xrightarrow{l}_{r} y_0 \mid\mid y_1} \quad comm(l_0, l_1) = l$$

Similar to the previous case, in the above specification **(p0)** and **(p1)** are commutative mirrors of each other. But for **(p2)** to be the commutative mirror of itself, we need commutativity of the communication merge $\mid$. Hence, we have to check the rule **(cp0)**, as well. Rule **(cp0)** is the commutative mirror of itself, if *comm* is a commutative function. For the other remaining rule, since $\|\!\lfloor$ is not a commutative operator, **(lp0)** only needs to be in tyft format which is indeed the case.

**Example 4.10 (Nondeterministic Choice: Standard)** Consider the following transition system specification for a nondeterministic choice operator:

$$\textbf{(c0)}\,\frac{x_0 \xrightarrow{l}_r y_0}{x_0 + x_1 \xrightarrow{l}_r y_0} \qquad \textbf{(c1)}\,\frac{x_1 \xrightarrow{l}_r y_1}{x_0 + x_1 \xrightarrow{l}_r y_1}$$

Then, following Theorem 4.6, we can derive that nondeterministic choice is a commutative operator: Both **(c0)** and **(c1)** are in tyft format and each is the commutative mirror of the other under the mapping $\hbar(x_0) = x_1$, $\hbar(x_1) = x_0$, $\hbar(y_1) = y_0$ and $\hbar(y_0) = y_1$.

The following example illustrates that allowing a premise to be mapped to the conclusion of another rule can be useful.

**Example 4.11 (Nondeterministic Choice: Non-Standard)** As an alternative to the transition system specification from the previous example, nondeterministic choice can also be defined by means of the following transition system specification.

$$\textbf{(c0)}\,\frac{x_0 \xrightarrow{l}_r y_0}{x_0 + x_1 \xrightarrow{l}_r y_0} \qquad \textbf{(c1)}\,\frac{x_1 + x_0 \xrightarrow{l}_r y_0}{x_0 + x_1 \xrightarrow{l}_r y_0}$$

Deduction rule **(c0)** can not be matched by deduction rule **(c0)** itself. Deduction rule **(c0)** is matched by deduction rule **(c1)** using the mapping $\hbar(x_0) = x_1$, $\hbar(x_1) = x_0$, and $\hbar(y_0) = y_0$. Observe that if we would not have allowed the premise of **(c1)** to match the conclusion of **(c0)**, then it would have been impossible to match **(c0)**. Hence, Theorem 4.6 could not have been applied here, although nondeterministic choice as specified by this transition system specification is commutative.

Rule **(c1)** is the SOS characterization of commutativity and henceforth one may expect that by adding this rule for a specific operator to any TSS, one should be able to make it commutative. It can be easily shown, using the comm-tyft format that this is indeed true. For any (binary) operator $f$, such a rule can play the role of a commutative mirror of all $f$-defining rules in the TSS by taking a mapping similar to the one above and hence making the TSS conform to the comm-tyft format.

The following example illustrates that we cannot relax the last constraint of the comm-tyft format to include $CC$-variants of the conclusion in the premises of the mirror rule.

**Example 4.12** Suppose that we slightly relax the last constraint of the comm-tyft format to the following:

$$\forall_{h \in H} \hbar(h) \in [H \cup \{c\}]_{cc}$$

Then, the commutativity result is jeopardized. The following counter-example shows this fact:

$$\textbf{(a)}\frac{}{a \xrightarrow{a}_r b} \qquad \textbf{(d)}\frac{x_0 \xrightarrow{l}_r y_0}{x_0 + x_1 \xrightarrow{l}_r y_0} \qquad \textbf{(d')}\frac{x_0 + x_1 \xrightarrow{l}_r y_0}{x_0 + x_1 \xrightarrow{l}_r y_0}$$

Suppose that the signature contains constants $a$ and $b$ and a binary function symbol $+$. The above transition system specification is in the relaxed version of the comm-tyft format with respect to the set $COMM = \{+\}$; all rules are in tyft format and **(d')** is the commutative mirror of **(d)** and itself under $\hbar(x_0) = x_1$, $\hbar(x_1) = x_0$ and $\hbar(y_0) = y_0$. However, it does not hold that $a + b \underline{\leftrightarrow} b + a$ since $a + b$ can make a transition due to **(d)** while $b + a$ cannot.

## 4.4 Possible Extensions

### 4.4.1 Tyxt Rules

According to Definition 3.6, rules in tyxt format are of the following form:

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\}}{x \xrightarrow{l}_r t}$$

Interesting enough, we can extend our comm-tyft format to allow for arbitrary rules in tyxt format. In [64], it has been already shown that tyxt format reduces to tyft format by copying the rule for all $(f, ar(f)) \in \Sigma$ and substituting variable $x$ by $f(\overrightarrow{x}_{ar(f)-1})$. So, tyxt rules do not harm the congruence property. They do not harm commutativity, either, since after such a copying procedure, each copied tyxt rule (for operators in the set $COMM$) is the commutative mirror of itself by taking the mapping $\hbar(x_i) = x_j$ and $\hbar(x_j) = x_i$ for any two arbitrary $0 \leq i < j < ar(f)$ and $\hbar(x) = x$ for all other variables.

### 4.4.2 Predicates and Negative Premises

Predicates are used to specify properties on process terms such as termination and divergence. By adding predicates to our set of formulae, the syntax of a deduction rule will be extended to the following forms:

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\} \quad \{P_j(t_j) | j \in J\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t} \qquad \frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\} \quad \{P_j(t_j) | j \in J\}}{P(f(\overrightarrow{x}_{ar(f)-1}))}$$

In [12], it is also shown that the above rules can be reduced to tyft format by introducing dummy transition relations for each predicate symbol, fresh variables (w.r.t. other variables in the deduction rule) in the target of the predicate formulae in the premises and dummy fresh constants (w.r.t. constants appearing in the signature) in the target of the predicate conclusions. The same trick works here, as well. Thus, to interpret our comm-tyft format in a setting with predicates, it suffices to consider predicates as sources of transitions. We can safely assume that the targets of such transitions satisfy our format (by taking the same dummy variables and constants in mirror rules).

In [61], it is shown that allowing for negative premises (of the form $t_i \not\rightarrow$) in a transition system specification may endanger the well-definedness of the induced transition relation. Several approaches have been proposed to deal with this problem, of which [59] provides an overview. Stratification (see Definition 3.9) is a measure defined on formulae which does not increase from conclusion to positive premises and decreases from conclusion to negative ones. If such a stratification exists, it has been shown in [27] that the induced transition relation is well-defined. Thus, by following the same approach we may accommodate negative premises in the comm-tyft format, too. All in all, by allowing for tyxt rules, negative premises and predicates, we get the expressive power of PANTH (*P*redicates *A*nd *NT*yft-ntyxt *H*ybrid) format of [134].

## 4.5 Conclusion

In this chapter, we defined a standard SOS format that guarantees that a number of commutativity axioms are sound with respect to strong bisimilarity (and all weaker notions of behavioral equivalence). Our standard format for commutativity, called comm-tyft, calls for so-called commutative mirror rules for each deduction rule defining a commutative operator. These mirror rules account for arbitrary switching of arguments and thus their existence is a sufficient condition for commutativity. The proposed standard format can help as a theoretical background for part of a toolkit assisting specifiers in defining operational semantics and proving meta-properties about their defined languages. We have tried the same method for more complicated frequently occurring axioms, such as associativity. It turns out that the result is an abstract representation of the proof for those axioms. In such a proof structure, one usually has to decompose the reason for a transition of $f(f(p,q),r)$ to transitions of $p$, $q$ and $r$ (by analyzing the proof structure to depth 2 and making several case distinctions based on the structure of the deduction rules) and then using the deduction rules in the TSS, compose these transitions again and prove the same transition (up to associativity of the target) for $f(p,f(q,r))$. The resulting format for associativity should contain all such analysis and case distinctions and thus is far from elegant.

# Chapter 5

# Structural Congruences

> "We must recover the element of quality in our traditional pursuit of equality."
>
> [Adlai Stevenson]

---

A summarized version of this chapter has appeared as: M.R. Mousavi, M.A. Reniers, Congruence for Structural Congruences, In V. Sassone ed., *Proceedings of the Eighth International Conference on Foundations of Software Science and Computation Structures (FOSSACS'05)*, Edinburgh, Scotland, UK, volume 3441 of Lecture Notes in Computer Science, pp. 47–62, Springer-Verlag, April 2005.

## 5.1 Introduction

Structural congruences were introduced in [88, 89] in the operational semantics specification of the $\pi$-calculus. There, structural congruences are a set of equations defining an equality and congruence relation on process terms. These equations are used as an addendum to transition system specifications (TSS's). The two specifications (structural congruences and TSS's) are linked using a deduction rule dedicated to the behavior of congruent terms, stating that if a process term can perform a transition, all congruent process terms can mimic the same behavior.

The combination of structural congruences and SOS rules may simplify SOS specifications and make them look more compact. They can also capture inherent (so-called *spatial*) properties of composition operators (e.g., commutativity, associativity and zero element). Perhaps, the latter has been the main reason for using them in combination with SOS. However, as we argue in this chapter, the interaction between the two specification styles is not as trivial as it seems. Particularly, well-definedness (i.e., existence and uniqueness of the induced transition relation) and well-behavedness (e.g., congruence of bisimilarity) meta-theorems for SOS do not carry over trivially to this mixed setting. As an interesting example, we show that the addition of structural congruences to a set of safe SOS rules (e.g., tyft rules) can put the congruence property of bisimilarity in jeopardy. This result shows that a standard congruence format cannot be used, as is, for the combination of structural congruences and SOS rules. As another example, we show that the well-definedness criteria defined in [27, 61] for SOS with negative premises do not necessarily suffice in the setting with structural congruences.

Three solutions can be proposed to deal with the aforementioned problems. The first is to avoid using structural congruences and use "pure" SOS specifications for defining operational semantics. In this approach, there is a conceptual distinction between the transition system semantics (as the model of the algebra) and the equational theory (cf. [13], for example). This way, one may lose the compactness and the intuitive presentation of the operational semantics, but in return, one will be able to benefit from the existing theories of SOS. This solution can be recommended as a homogenous way of specifying semantics. The second solution is to use structural congruences in combination with SOS rules and prove the well-behavedness theorems (e.g., well-definedness of the semantics and congruence of the notion of equality) manually. By taking this solution, all the tedious proofs of congruence, as a typical example, have to be done manually and re-done or adapted in the case of each single change in the syntax and semantics. Thus, this solution does not seem promising at all. The third solution is to extend meta-theorems of SOS to this mixed setting. In this chapter, we pursue the third solution.

The rest of this chapter is structured as follows. In Section 5.2, we review the related work. Subsequently, Section 5.3 is devoted to accommodating structural congruences in the SOS framework. We propose a number of SOS interpretations for structural congruences and compare them formally. Congruence is by itself an interesting and essential property for bisimilarity (as an equivalence). Furthermore, it turns out that it plays a role in relating different interpretations of structural congruences. Thus, in Section 5.4, we study structural congruences from the congruence point of view. There, we propose a syntactic format for structural congruences that induces congruence of strong bisimilarity. We show, by several abstract counter-examples, that our syntactic format cannot be relaxed in any obvious way and dropping any of the syntactic restrictions may destroy the congruence property in general. In Section 5.5, we extend our format to cater for SOS rules with negative premises. To illustrate our congruence format with a concrete example, in Section 5.6, we apply it to a CCS-like process algebra. Finally, Section 5.7 concludes the chapter and points out possible extensions of our work. For the rest of this chapter, we assume familiarity with TSS's with constant labels (Definition 3.1), congruence (Definition 3.3) and strong bisimilarity (Definition 3.4).

## 5.2 Related Work

This section is concerned with the origins of and recent developments concerning structural congruences and their relationship with Structural Operational Semantics.

Structural congruences find their origin in the chemical models of computation [18]. The Chemical Abstract Machine (Cham) of [22] is among the early instances of such models. In Cham, parallel agents are modelled by molecules floating around in a chemical solution. The solution is constantly stirred using a *magical mechanism*, in the spirit of the *Brownian motion* in chemistry, that allows for possible contacts among reacting molecules.

Inspired by the magical mechanism of Cham, structural congruences were introduced in [88, 89] in the semantic specification of the $\pi$-calculus. Before that, an equivalent semantics of the $\pi$-calculus had been presented in [91], in terms of "pure" SOS rules. As stated in [92], structural congruences were also inspired by a curious difference between lambda-calculi and process calculi; in lambda-calculi, interacting terms are always placed adjacently in the syntax, while in process calculi, interacting agents may be dispersed around the process term due to syntactic restrictions. Thus, part of the idea is to bring interacting terms together by considering terms modulo structural changes. However, the application of structural congruences is not restricted to this concept. Structural congruences have also been used to define the semantics of new operators in terms of previously defined ones (e.g., defining the semantics of the parallel replication operator in terms of parallel composition, i.e., $!x \doteq x \mid\mid !x$, in [88, 89] and Section 5.6 of this chapter). This is similar in essence to the concept of *definitional extensions* in [9].

Before that, in [85], structural congruences were presented for a subset of Calculus of Communicating Systems (CCS) under the name *flow-algebra* (rules). However, the use of structural congruences in [85] is essentially different from [88, 89]. In [85], structural congruences are a set of equalities on an algebraic signature. Similar to [85], in the ACP style of process algebra (e.g., in [13]), there is a conceptual distinction between such structural rules, as the equational theory of the algebra and the (transition system) semantics, as its model. From this point of view, SOS is a means to define the transition systems semantics and thus is not directly connected to the structural rules. Structural rules are to be proved sound (and possibly complete) with respect to the defined semantics and the particular notion of equivalence. In contrast, in [88, 89], they are used as a means to define or augment the operational semantics of a language. The present chapter is concerned with the structural congruences in the sense of [88, 89].

The practice of using structural congruences for the specification of operational semantics, à la Milner [88], has continued since then. See [37], [39] and [44] for recent examples in defining operational semantics for Mobile Ambients, GAMMA and its coordination language, and the $\pi$-calculus, respectively.

There have been a number of recent works devoted to the fundamental study of formal semantics with structural congruences. Among these, we can refer to [**?**, 77, 78, 116, 117]. The lack of a congruent notion of bisimilarity for the semantics of the $\pi$-calculus has been known since [88] (which is not only due to structural congruences), but most attempts (e.g., [77, 78, 92, 116, 117]) were focused on deriving a suitable transition system (e.g., *contexts as labels* approach of [77, 78]) or a notion of equivalence (e.g., barbed congruence of [92]) that induce a congruence. The works of [77, 78, 116, 117] deviate from the traditional interpretation of SOS deduction rules and establish a new semantic framework close to the reduction (reaction) rules of lambda calculus [69]. Arguably, this semantic framework is neither necessarily *structural*, i.e., following precisely the structure of syntax, nor *structured*, i.e., guaranteeing well-behavedness criteria such as congruence. For example, in [117], it is emphasized that the relation between this framework and the known congruence results for SOS remains to be established and the present chapter realizes this goal (at least partially). Thus, compared to the above approaches, we take a different angle to the problem, that is, to characterize the set of specifications that induce a reasonable transition relation in its commonly accepted meaning. In [36], a similar approach is used to derive congruence formats for tile bisimilarity [51]. The syntactic congruence format of [36] is more restricted than ours but our results are incomparable since our notions of bisimilarity differs.

To this end, we transfer the SOS meta-theorems concerning congruence of strong bisimilarity and well-definedness of the semantics to the setting with structural congruences. For simplicity, we consider TSS's in tyft format with a single transition relation. Then, we extend our framework with negative premises and study the consequences of this extension.

## 5.3 Structural Congruences: Three Operational Interpretations

Structural congruences consist of a set of equations on open terms, denoted by $t \equiv t'$ on a given signature. As interpreted by [88], these equations induce a congruence (and equivalence) relation on closed terms. Then, they are connected to an SOS specification by means of a special deduction rule, stating that if a term can perform a transition, its congruent terms can mimic the same transition.

We take this interpretation as the original and intuitive meaning of structural congruences and give it a formal meaning in Section 5.3.1. Moreover, we present two alternative interpretations in Sections 5.3.2 and 5.3.3. In Section 5.3.2, we introduce the notation $\equiv$ as a new transition relation in the TSS. This way, equations of structural congruence, naturally turn into SOS axioms. Section 5.3.3 considers structural congruences as specifications of bisimilar terms. Thus, it adds two deduction rules for each equation, stating that if one side of a structural congruence equation can perform a transition, the other side can perform the same transition and vice versa.

Informally speaking, the first interpretation is the closest to the intuition behind structural congruences but as we move on to the second and the third, the resulting interpretation fits more in the TSS framework. While for the first interpretation, the notions of proof and provable transitions have to be adapted, for the second we only have to add a new transition relation and a number of deduction rules (thus, only a syntactic manipulation of TSS's) and in the third, even structural congruences do not show up in the TSS and just new deduction rules have to be added.

We also present a formal comparison of the three interpretations of structural congruences. In particular, we show that the first and the second interpretations, despite their different presentations, coincide. However, the third interpretation only coincides with the first two if the original TSS is in tyft format and furthermore bisimilarity is a congruence. In fact, the congruence condition turns out to be tricky, as structural congruences may jeopardize it even if they are added to a set of tyft rules (which by themselves guarantee the congruence). This sets the scene for Section 5.4, where we define syntactic criteria on structural congruences to derive congruence for strong bisimilarity.

### 5.3.1 External Interpretation

Structural congruences $sc$ on a set of variables $V$ and a signature $\Sigma$ consist of a set of equations of the form $t \equiv t'$, where $t, t' \in \mathcal{T}$. They induce a structural congruence relation on closed terms, as defined below.

**Definition 5.1 (Structural Congruence Relation)** A structural congruence relation induced by structural congruences $sc$ on signature $\Sigma$, denoted by $\equiv_{sc}$, is the minimal relation satisfying the following constraints:

1. $\forall_{p \in \mathcal{C}}\ p \equiv_{sc} p$ (reflexivity);
2. $\forall_{p,q,r \in \mathcal{C}}\ (p \equiv_{sc} q \wedge q \equiv_{sc} r) \Rightarrow p \equiv_{sc} r$ (transitivity);
3. $\forall_{(f,ar(f)) \in \Sigma} \forall_{0 \leq i < ar(f)} \forall_{\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1} \in \mathcal{C}}\ \overrightarrow{p}_{ar(f)-1} \equiv_{sc} \overrightarrow{q}_{ar(f)-1} \Rightarrow f(\overrightarrow{p}_{ar(f)-1}) \equiv_{sc} f(\overrightarrow{q}_{ar(f)-1})$ (congruence);
4. $\forall_{\sigma : V \to \mathcal{C}} \forall_{t,t' \in \mathcal{T}}\ (t \equiv t') \in sc \Rightarrow (\sigma(t) \equiv_{sc} \sigma(t') \wedge \sigma(t') \equiv_{sc} \sigma(t))$ (structural congruences).

It can easily be checked that $\equiv_{sc}$ is symmetric and thus, alternatively, $\equiv_{sc}$ is the smallest congruence relation satisfying structural congruences on closed terms.

In the rest of this chapter, we assume that the structural congruences have the same signature as the TSS they are added to.

To link structural congruences to a TSS, a special rule is used, which we call *the structural congruence rule.*

**Definition 5.2 (The Structural Congruence Rule [88])** The particular rule schema of the following form (which is in fact a set of deduction rules for all $l \in L$) is called the structural congruence rule.

$$\textbf{(struct)}\frac{x \equiv y \quad y \xrightarrow{l} y' \quad y' \equiv x'}{x \xrightarrow{l} x'}(l \in L)$$

Consider a TSS $tss = (\Sigma, V, L, \{\to\}, D)$ and structural congruences $sc$ on the same signature. The extension of $tss$ with $sc$, denoted by $tss \cup \{\textbf{(struct)}\}$, is defined by the tuple $(\Sigma, V, L, \{\to\}, D \cup \{\textbf{(struct)}\})$.

There remains a problem concerning Definition 5.2, namely, the structural congruence rule does not fit within the notion of a deduction rule as defined in Definition 3.1 since structural congruences (appearing in the premises) do not fit the definition of formulae (e.g., in Definition 3.1) per se. In other words, $x \equiv y$ is only a syntactic notation and we have not assigned any meaning to it, as yet. In fact, this subsection and the following two are concerned with different interpretations of the symbol $\equiv$. In this subsection, we do not interpret $\equiv$ directly, but rather exploit the structural congruence relation to extend the notion of proof. Syntactically, we allow for deduction rules of the following form:

$$\frac{\{\chi_i | i \in I\} \quad \{t_j \equiv t'_j | j \in J\}}{\chi}$$

where $\chi$ and $\chi_i$'s are positive transition formulae as defined before (in Definition 3.1) and $t_j$ and $t'_j$ are terms from the signature. This rule format, easily accommodates the structural congruence rule. Then, we extend the notion of provable transitions to the following notion:

**Definition 5.3 (Provable Transitions: Extended)** A *proof* of a closed formula $\phi$ (in an extended TSS $tss \cup \{\textbf{(struct)}\}$) is a well-founded upwardly branching tree of which the nodes are labelled by closed formulae such that

- the root node is labelled by $\phi$, and
- if $\psi$ is the label of a node $q$ and $\{\psi_i \mid i \in I\}$ is the set of labels of the nodes directly above $q$, then there is a deduction rule $\dfrac{\{\chi_i \mid i \in I\} \quad \{t_j \equiv t'_j | j \in J\}}{\chi}$ (in $tss \cup \{\textbf{(struct)}\}$) and a substitution $\sigma$ such that $\sigma(\chi) = \psi$, for all $i \in I$, $\sigma(\chi_i) = \psi_i$, and for all $j \in J$, $\sigma(t_j) \equiv_{sc} \sigma(t'_j)$.

We re-use the same notations for provability of formulae in the extended setting.

The following lemma shows that in our new framework, congruent (closed) terms are indeed bisimilar.

**Lemma 5.4** Consider a TSS $tss$ and structural congruences $sc$, and the bisimilarity relation $\underline{\leftrightarrow}$ with respect to $tss \cup \{\textbf{(struct)}\}$. It holds that $\equiv_{sc} \subseteq \underline{\leftrightarrow}$.

*Proof.* Immediate, from the definition of bisimilarity and the structural congruence rule by taking $\equiv_{sc}$ as a bisimulation relation. ⊠

We may simplify Definition 5.1 by adding a simpler rule to the TSS. The following (simpler) rule only allows for congruent terms to be replaced in the source of the transition.

$$\textbf{(struct')}\frac{x \equiv y \quad y \xrightarrow{l} y'}{x \xrightarrow{l} y'}(l \in L)$$

Suppose that we replace rule **(struct)** in Definition 5.1 with **(struct')** and denote the new TSS by $tss \cup \{\textbf{(struct')}\}$. It turns out that this slight simplification amounts to a TSS that is equal to $tss \cup \{\textbf{(struct)}\}$ up to bisimilarity, if the original $tss$ is in tyft format and bisimilarity is a congruence. We first define what coincidence up to bisimilarity means and then formulate and prove the result mentioned above.

Bisimilarity is an equivalence relation and thus partitions the set of process terms into equivalence classes. We use this concept to define equality of transition relations and TSS's up to bisimilarity.

**Definition 5.5 (Inclusion and Equality up to Bisimilarity)** A transition relation $\rightarrow_0 \subseteq \mathcal{C} \times L \times \mathcal{C}$ is included in $\rightarrow_1 \subseteq \mathcal{C} \times L \times \mathcal{C}$ up to bisimilarity if and only if for all transitions $p \xrightarrow{l}_0 p'$, there exists a closed term $p''$ such that $p \xrightarrow{l}_1 p''$ and $p' \underline{\leftrightarrow}_1 p''$ (where $\underline{\leftrightarrow}_1$ is bisimilarity with respect to $\rightarrow_1$). Two transition relations are equal up to bisimilarity if and only if the inclusions hold in both directions.

Two TSS's are called equal if they induce the same transition relations. They are called equal up to bisimilarity if their induced transition relations are equal up to bisimilarity.

Note that inclusion (equality) of transition relations implies inclusion (equality) up to bisimilarity but not vice versa. Inclusion (up to bisimilarity) is reflexive and transitive and equality (up to bisimilarity) is an equivalence relation.

**Corollary 5.6** Suppose that $\rightarrow_0 \subseteq \mathcal{C} \times L \times \mathcal{C}$ is equal to $\rightarrow_1 \subseteq \mathcal{C} \times L \times \mathcal{C}$ up to bisimilarity; then for all closed terms $p, q \in \mathcal{C}$, $p \underline{\leftrightarrow}\, q$ with respect to $\rightarrow_0$ if and only if $p \underline{\leftrightarrow}\, q$ with respect to $\rightarrow_1$.

*Proof.* Immediate consequence of Definition 5.5. ⊠

A result that comes in handy is that congruence of bisimilarity with respect to transition relations equal up to bisimilarity coincides.

**Corollary 5.7** If transition relations $\rightarrow_0$ and $\rightarrow_1$ are equal up to bisimilarity then bisimilarity with respect to $\rightarrow_0$ is a congruence if and only if bisimilarity with respect to $\rightarrow_1$ is a congruence.

*Proof.* Immediate result of Corollary 5.6. ⊠

The following two Lemmas establish the equality of $tss \cup \{$**(struct)**$\}$ and $tss \cup \{$**(struct')**$\}$ up to bisimilarity.

**Lemma 5.8** For $tss$ in well-founded $\mathsf{tyft}$ format and structural congruences $sc$, if bisimilarity with respect to $tss \cup \{$**(struct')**$\}$ is a congruence, then the transition relation induced by $tss \cup \{$**(struct)**$\}$ is included in the transition relation induced by $tss \cup \{$**(struct')**$\}$ up to bisimilarity.

*Proof.* Let $\rightarrow_0$ and $\rightarrow_1$ be the two transition relations induced by $tss \cup \{$**(struct)**$\}$ and $tss \cup \{$**(struct')**$\}$, respectively. We have to show that $\rightarrow_0$ is included in $\rightarrow_1$ up to bisimilarity. Instead, we show that if $tss \cup \{$**(struct)**$\} \vdash p \xrightarrow{l}_0 p'$, for arbitrary closed terms $p$ and $p'$ and label $l$, then $tss \cup \{\textbf{(struct')}\} \vdash p \xrightarrow{l}_1 p''$ where $p'' \equiv_{sc} p'$. Then, the thesis follows from Lemma 5.4.

We prove this by an induction on the depth of the proof tree resulting in this transition (see Definition 5.3).

If transition $tss \cup \{\textbf{(struct)}\} \vdash p \xrightarrow{l}_0 p'$ has a proof of depth 1, then it should be due to an axiom in $tss$ and a substitution $\sigma$ (transitions due to **(struct)** have a proof depth of at least 2 since they always have a transition in their premises). Then, it immediately follows from the same axiom (using substitution $\sigma$) that there is a proof for $p \xrightarrow{l}_1 p'$ in $tss \cup \{\textbf{(struct')}\}$ (and $p' \equiv_{sc} p'$ holds trivially).

For the induction step, suppose that the transfer condition holds for all transitions with a proof of depth $n-1$ or less. Consider transition $p \xrightarrow{l}_0 p'$ which has a proof of depth $n$. If the transition is due to a rule in $tss$ of the following form (for an arbitrary $n$-ary function symbol $f$):

$$\frac{\{t_i \xrightarrow{l_i} y_i | i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} t}$$

then there exists a substitution $\sigma$ such that $\sigma(x_i) = p_i$ $(0 \leq i < ar(f))$, $\sigma(t) = p'$, $p = f(\overrightarrow{p}_{ar(f)-1})$ and $tss \cup \{\textbf{(struct)}\} \vdash \sigma(t_i) \xrightarrow{l_i} \sigma(y_i)$ for all $i \in I$.

Since we assumed acyclicity of the variable dependency graph, we can define a rank, $rank(x)$, for each variable $x$, as the maximum length of a backward chain starting from $x$ in the variable dependency graph. The rank of a premise is the rank of its target variable. Then, for each $x \in vars(t_i)$ of each premise $t_i \xrightarrow{l_i} y_i$ of the deduction rule, it holds that $rank(x) < rank(y_i)$.

Take $Y = \{y_i | i \in I\}$. We define a new substitution $\sigma'$ such that for all $x \in V \setminus Y$, $\sigma'(x) \doteq \sigma(x)$. Note that thus far this substitution is not defined for variables in $Y$. We extend the definition while proving, by induction on the rank of a premise $r$, the preservation of three essential properties:

1. $\sigma(t_i) \equiv_{sc} \sigma'(t_i)$;
2. $\sigma'(t_i) \xrightarrow{l_i}_1 \sigma'(y_i)$;
3. $\sigma(y_i) \equiv_{sc} \sigma'(y_i)$.

Take any premise, say some $i \in I$, that has a minimal rank of which the target variable $y_i$ is not defined under $\sigma'$. The source term of this premise ($t_i$) is fully interpreted under $\sigma'$ (i.e., $\sigma'(t_i)$ is a closed term) and it follows from Lemma 4.7 (since $\equiv_{sc}$ is a congruence) and the above constraints that $\sigma(t_i) \equiv_{sc} \sigma'(t_i)$. Since $\sigma(t_i) \xrightarrow{l}_0 \sigma(y_i)$ has a proof of depth $n-1$ or less, it follows from the induction hypothesis that $\sigma'(t_i) \xrightarrow{l}_1 p'_i$ and $\sigma(y_i) \equiv_{sc} p'_i$ for some $p'_i$. Then take $\sigma'(y_i) \doteq p'_i$. Using this inductive procedure, we complete the definition of $\sigma'$ for all $y_i$'s and thus for all variables $x$, maintaining the property $\sigma(x) \equiv_{sc} \sigma'(x)$. Then, using the same rule, we have a proof for $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) \xrightarrow{l}_1 \sigma'(t)$, or $p \xrightarrow{l}_1 \sigma'(t)$ and it holds that $p' = \sigma(t) \equiv_{sc} \sigma'(t)$.

If the transition is due to the congruence rule of the following form

$$\textbf{(struct)}\frac{x \equiv y \quad y \xrightarrow{l} y' \quad y' \equiv x'}{x \xrightarrow{l} x'}$$

then there is a substitution $\sigma$ such that for some $q$ and $q'$, $\sigma(x) = p$, $\sigma(y) = q$, $\sigma(y') = q'$ and $\sigma(x') = p'$ and it holds that $p \equiv_{sc} q$, $p' \equiv_{sc} q'$ and $q \xrightarrow{l}_0 q'$. By applying the induction hypothesis on $q \xrightarrow{l}_0 q'$, we get $q \xrightarrow{l}_1 q''$ for some $q''$ such that $q'' \equiv_{sc} q'$ and by symmetry and transitivity of $\equiv_{sc}$, we get $p' \equiv q''$. Thus, by using deduction rule

$$\textbf{(struct')}\frac{x \equiv y \quad y \xrightarrow{l} y'}{x \xrightarrow{l} y'}$$

and a substitution $\sigma'$ such that $\sigma'(x) = p$, $\sigma'(y) = q$, $\sigma'(y') = q''$, we have a proof for $p \xrightarrow{l}_1 q''$ and we have already shown that $p' \equiv_{sc} q''$. This concludes the proof. ⊠

The above lemma cannot be generalized to arbitrary TSS's (not in the tyft format). The following example illustrates this fact.

**Example 5.9** Consider the following structural congruence $sc$ and TSS $tss$.

$$a \equiv f(b)$$

$$\textbf{(a)}\frac{}{a \xrightarrow{l_0} a} \qquad \textbf{(xb)}\frac{x \xrightarrow{l_0} f(b)}{x \xrightarrow{l_0} b}$$

Suppose that the common signature contains $a$ and $b$ as constants and $f$ as a unary function symbol. Since $a \equiv f(b)$, using **(a)** and **(struct)**, we can prove that $a \xrightarrow{l_0} f(b)$ and hence it follows from **(xb)** that $a \xrightarrow{l_0} b$. However, using **(struct')** we are not able to prove $a \xrightarrow{l_0} b$ since $a \xrightarrow{l_0} f(b)$ is not provable anymore. Furthermore, we cannot prove $a \xrightarrow{l_0} p$ for any $p \underline{\leftrightarrow} b$. This shows that the two TSS's $tss \cup \{\textbf{(struct)}\}$ and $tss \cup \{\textbf{(struct')}\}$ are not equal up to bisimilarity.

Inclusion of the transition relation induced by $tss \cup \{\textbf{(struct')}\}$ in $tss \cup \{\textbf{(struct)}\}$, however, does not require any assumption about the TSS.

**Lemma 5.10** For an arbitrary TSS $tss$ and structural congruences $sc$, the transition relation induced by $tss \cup \{\textbf{(struct')}\}$ is a subset of the transition relation induced by $tss \cup \{\textbf{(struct)}\}$.

*Proof.* Straightforward from an induction on the depth of the proof of transitions in $tss \cup \{\textbf{(struct')}\}$. Note that $\equiv_{sc}$ is reflexive and thus any transition provable from rule **(struct')** is also provable from **(struct)**. ⊠

We are not able to reproduce the results of Theorem 3.7 (concerning congruence for the tyft format) in the extended setting with structural congruences. In fact, adding structural congruences to a set of tyft rules does not preserve the congruence property of bisimilarity. The following counter-example shows this fact.

**Example 5.11** Consider the following structural congruence and TSS. The common signature is assumed to have $a$ and $b$ as constants and $f$ as a unary operator.

$$a \equiv f(b)$$

$$\textbf{(a)}\frac{}{a \xrightarrow{l_0} a} \qquad \textbf{(b)}\frac{}{b \xrightarrow{l_0} a}$$

In the above specification, both $a$ and $b$ can perform an $l_0$ transition to $a$ due to rules **(a)** and **(b)**, respectively. On one hand, using Definition 5.1, $a$ is only congruent to itself and $f(b)$. On the other hand, $b$ is only congruent to itself. Since $f(b)$ cannot perform any new transition, neither $a$ nor $b$ can perform any other transition due to **(struct)**. Thus, to this end, we have $a \underline{\leftrightarrow} b$. However, it does not hold that $f(a) \underline{\leftrightarrow} f(b)$ since $f(a)$ cannot perform any transition (it is only congruent to $f(f(b))$ which cannot perform any transition either), but $f(b)$ can perform an $l_0$ transition to $a$ (using **(struct)** since it is congruent to $a$). This shows that bisimilarity is not a congruence in the above TSS, despite the fact that the original TSS is in tyft format.

Several other counter-examples of violating the congruence property by structural congruences are presented in the remainder of this paper.

### 5.3.2 Transition Relation Interpretation

The second interpretation of structural congruences, considers $\equiv$ as a new transition relation in the TSS. Thus, structural congruence equation $t \equiv t'$ is interpreted as the following SOS axiom:

$$\textbf{(tt')}\frac{}{t \equiv t'}$$

To be more precise $\equiv$ is defined as the pair $(\rightsquigarrow, \square)$ where $\rightsquigarrow$ is a fresh transition relation and $\square$ is a fresh label. (In fact, freshness of either of the two suffices.)

Since transition relations are directed, we have to add another deduction rule to account for the natural symmetry in $\equiv$:

$$\textbf{(t't)}\frac{}{t' \equiv t}$$

Also, to account for reflexivity, transitivity and congruence of $\equiv$, we have to add the following rules to the TSS:

$$\textbf{(refl)}\frac{}{x \equiv x} \qquad \textbf{(trans)}\frac{x \equiv y \quad y \equiv z}{x \equiv z}$$

$$\textbf{(congf)}\frac{\{x_i \equiv y_i \mid 0 \leq i < ar(f)\}}{f(\overrightarrow{x}_{ar(f)-1}) \equiv f(\overrightarrow{y}_{ar(f)-1})} \quad \text{(for all } (f, ar(f)) \in \Sigma)$$

Then, the structural congruence rule

$$\textbf{(struct)}\frac{x \equiv y \quad y \xrightarrow{l} y' \quad y' \equiv x'}{x \xrightarrow{l} x'}$$

fits very well in the definition of a TSS with constant labels (Definition 3.1) since $x \equiv y$ and $x' \equiv y'$ are now valid formulae.

We summarize the new interpretation of structural congruences in the following definition.

**Definition 5.12** (Structural Congruences as a Transition Relation) The interpretation of structural congruences $sc$ on signature $\Sigma$ with a TSS $tss = (\Sigma, V, L, \{\rightarrow\}, D)$ is a new TSS $tss \cup \langle\!\langle sc \rangle\!\rangle \doteq (\Sigma, V, L \cup \{l_n\}, \{\rightarrow, \rightarrow_n\}, D \cup \langle\!\langle sc \rangle\!\rangle)$, where $\langle\!\langle sc \rangle\!\rangle$ is defined as follows:

$$\langle\!\langle sc \rangle\!\rangle \doteq \quad \{\textbf{(refl)}, \textbf{(trans)}, \textbf{(struct)}\} \cup \{\textbf{(congf)} \mid (f, ar(f)) \in \Sigma\} \cup$$

$$\left\{ \begin{array}{l|l} \textbf{(tt')}\dfrac{}{t \equiv t'}, & \\ & (t \equiv t') \in sc \\ \textbf{(t't)}\dfrac{}{t' \equiv t} & \end{array} \right\}$$

where **(refl)**, **(trans)**, **(congf)** and **(struct)** are the reflexivity, transitivity, congruence and structural congruence rules, respectively, as defined before.

We are now in the position to rephrase Lemma 5.4 for the new interpretation. Namely, we can now prove that for all provable (transitions) $p \equiv q$, $p$ and $q$ are bisimilar.

**Lemma 5.13** Consider a TSS $tss$ and structural congruences $sc$. Take $\Sigma$ to be the common signature of $sc$ and $tss$. Then, for all $p, q \in \mathcal{C}$ such that $tss \cup \langle\langle sc \rangle\rangle \vdash p \equiv q$, we have $tss \cup \langle\langle sc \rangle\rangle \vdash p \leftrightarroweq q$.

*Proof.* Take $R$ to be the set of all pairs of closed terms $(p, q)$ such that $p \equiv q$ is a provable transition. Consider an arbitrary pair $(p, q) \in R$; we have to prove that for all $p' \in \mathcal{C}$ and for all transitions of $p$ (both transitions of the form $p \xrightarrow{l} p'$ and $p \equiv p'$) there exists a $q'$ such that $q$ can make the same transitions to $q'$ and $(p', q') \in R$ (and vice versa which is symmetric to this case).

1. If $tss \cup \langle\langle sc \rangle\rangle \vdash p \xrightarrow{l} p'$, since $(p, q) \in R$ and $R$ is symmetric by construction, $(q, p) \in R$. Thus, $tss \cup \langle\langle sc \rangle\rangle \vdash q \equiv p$. Hence, it follows from **(struct)** that $tss \cup \langle\langle sc \rangle\rangle \vdash q \xrightarrow{l} p'$ using premises $q \equiv p$, $p \xrightarrow{l} p'$ and $p' \equiv p'$ (the last statement follows from rule **(refl)**).
2. If $tss \cup \langle\langle sc \rangle\rangle \vdash p \equiv p'$, we already know that $tss \cup \langle\langle sc \rangle\rangle \vdash q \equiv p$ (see the previous item) and thus it follows from **(trans)** that $tss \cup \langle\langle sc \rangle\rangle \vdash q \equiv p'$ and again $(p', p') \in R$ due to **(refl)**.

⊠

To compare the two interpretations given up to now, we first observe that the congruent classes of closed terms coincide for those.

**Lemma 5.14** For two closed terms $p$ and $q$, $p \equiv_{sc} q$ if and only if $p \equiv q$ is provable from $tss \cup \langle\langle sc \rangle\rangle$.

*Proof.* By straightforward inductions on the structure of $\equiv_{sc}$ and depth of the proof for $p \equiv q$ in $tss \cup \langle\langle sc \rangle\rangle$. ⊠

Using this lemma, we can easily show that the transition relations $\rightarrow$ induced by the two interpretations coincide.

**Theorem 5.15** For arbitrary closed terms $p$ and $p'$ and arbitrary label $l$, $p \xrightarrow{l} p'$ is provable from $tss \cup \{\textbf{(struct)}\}$ if and only if it is provable from $tss \cup \langle\langle sc \rangle\rangle$.

*Proof.* By a straightforward induction on the depth of the proofs for the transition $p \xrightarrow{l} p'$ and by using Lemma 5.14. ⊠

Also, it can be easily proven that replacing **(struct)** with **(struct')** in the definition of $\langle\!\langle sc \rangle\!\rangle$ results in an equal transition relation up to bisimilarity, provided that the necessary conditions of Lemma 5.8 hold. We dispense with repeating the lemma and the proof. We conclude this subsection by emphasizing that although the first and the second interpretations have different presentations, they are formally equal.

### 5.3.3 Bisimilarity Interpretation

An alternative way of interpreting structural congruences is to say that congruent terms should be able to mimic each others' transitions. In other words, the equation $t \equiv t'$ is interpreted as $\sigma(t) \underline{\leftrightarrow} \sigma(t')$ for all closed terms $\sigma(t)$ and $\sigma(t')$. To realize this interpretation in terms of SOS, we define a pair of rules for each equation (and for each label and transition relation), to prove all transitions of one side for the other side and vice versa. This concept is formally defined as follows.

**Definition 5.16 (Structural Congruences as Bisimilarity)** Consider structural congruences $sc$ on signature $\Sigma$ and a TSS $tss = (\Sigma, V, L, \{\rightarrow\}, D)$. We define a new TSS $tss \cup [\![sc]\!] \doteq (\Sigma, V, L, \{\rightarrow\}, D \cup [\![sc]\!])$, where $[\![sc]\!]$ is the SOS interpretation of $sc$, defined as follows:

$$[\![sc]\!] \doteq \left\{ \begin{array}{c|c} \textbf{(btt')}\dfrac{t \xrightarrow{l} y}{t' \xrightarrow{l} y}, & \\ & (t \equiv t') \in sc \\ \textbf{(bt't)}\dfrac{t' \xrightarrow{l} y}{t \xrightarrow{l} y} & \end{array} \right\}$$

In the above definition, $y$ is a fresh variable (i.e., $y \notin vars(t)$ and $y \notin vars(t')$).

We cannot present a direct proof for Lemma 5.4 (or similarly, Lemma 5.14) in this setting since $p \equiv_{sc} q$ (or $p \equiv q$) does not have any clear semantic counterpart in this interpretation. However, we can indirectly prove a similar result as of Lemmas 5.4 and 5.14. Namely, we can show that if $p \equiv_{sc} q$ (thus, $tss \cup \langle\!\langle sc \rangle\!\rangle \vdash p \equiv q$) , then $tss \cup [\![sc]\!] \vdash p \underline{\leftrightarrow} q$ provided that bisimilarity is a congruence. Next, we prove a lemma that establishes the aforementioned fact.

**Lemma 5.17** Consider a TSS $tss$ and structural congruences $sc$. If $p \equiv_{sc} q$ and bisimilarity w.r.t. $tss \cup [\![sc]\!]$ is a congruence, then it holds that $tss \cup [\![sc]\!] \vdash p \underline{\leftrightarrow} q$.

*Proof.* We prove the lemma by an induction on the structure of $\equiv_{sc}$:

1. Reflexivity: If $p \equiv_{sc} q$ is due to reflexivity, then the lemma follows trivially.

2. Transitivity: If $p \equiv_{sc} q$ is due to transitivity, then there exists a closed term $s$ such that $p \equiv_{sc} s$ and $s \equiv_{sc} q$. Then, according to the induction hypothesis $tss \cup [\![sc]\!] \vdash p \leftrightarrow s$ and $tss \cup [\![sc]\!] \vdash s \leftrightarrow q$ and since $\leftrightarrow$ is transitive $tss \cup [\![sc]\!] \vdash p \leftrightarrow q$.

3. Structural congruences: Then there is an equation $t \equiv t'$ (or similarly $t' \equiv t$, which is symmetric to this case) and a substitution $\sigma$ such that $\sigma(t) = p$ and $\sigma(t') = q$. It follows from Definition 5.16 that there is a rule **(btt')** in $tss \cup [\![sc]\!]$ of the following form:

$$\textbf{(btt')}\frac{t \xrightarrow{l} y}{t' \xrightarrow{l} y}(l \in L)$$

   Then, suppose that $p \xrightarrow{l} p'$ for an arbitrary $l$ and $p'$, by taking $\sigma'(x) \doteq \sigma(x)$ for $x \neq y$ and $\sigma'(y) \doteq p'$, we can derive $tss \cup [\![sc]\!] \vdash q \xrightarrow{l} p'$ using the above rule and substitution $\sigma'$ ($p' \leftrightarrow p'$ holds trivially).

4. Congruence: If $p \equiv_{sc} q$ is due to congruence, then, $p = f(\overrightarrow{p}_{ar(f)-1})$, $q = f(\overrightarrow{q}_{ar(f)-1})$ and $\overrightarrow{p}_{ar(f)-1} \equiv_{sc} \overrightarrow{q}_{ar(f)-1}$. It follows from the induction hypothesis that $tss \cup [\![sc]\!] \vdash \overrightarrow{p}_{ar(f)-1} \leftrightarrow \overrightarrow{q}_{ar(f)-1}$ and since bisimilarity is a congruence $tss \cup [\![sc]\!] \vdash p = f(\overrightarrow{p}_{ar(f)-1}) \leftrightarrow f(\overrightarrow{p}_{ar(f)-1}) = q$.

⊠

To compare this interpretation with the previous two interpretations, it suffices to compare it with one of them (as they are formally proved equal). Thus, we compare this interpretation with the first one. Next, we show that the transitions introduced by this interpretation are included in the transition relation induced by the first one.

**Theorem 5.18** For arbitrary closed terms $p$ and $p'$ and arbitrary label $l$ if $tss \cup [\![sc]\!] \vdash p \xrightarrow{l} p'$ then $tss \cup \{\textbf{(struct)}\} \vdash p \xrightarrow{l} p'$.

*Proof.* By an induction on the proof for $tss \cup [\![sc]\!] \vdash p \xrightarrow{l} p'$. For the induction basis, if the proof has depth one then it is due to an axiom in $tss$ and a substitution $\sigma$. Using the same axiom and the same substitution, we can derive that $tss \cup \{\textbf{(struct)}\} \vdash p \xrightarrow{l} p'$.

For the induction step, suppose that the theorem holds for all formulae with a proof of depth $n - 1$ or less and suppose that $tss \cup [\![sc]\!] \vdash p \xrightarrow{l} p'$ is due to a proof of depth $n$.

If the last deduction rule in the proof tree is in $tss$ then since all the premises of the rule have a proof of depth $n-1$ or less, they are all provable in $tss \cup \{\textbf{(struct)}\}$. Thus, using the same rule and same substitution, we have $tss \cup \{\textbf{(struct)}\} \vdash p \xrightarrow{l} p'$.

If the last deduction rule has the following form:

$$\textbf{(btt')}\frac{t \xrightarrow{l} y}{t' \xrightarrow{l} y}(l \in L)$$

then, there exists a substitution $\sigma$ such that $\sigma(t') = p$ and $\sigma(y) = p'$. According to Definition 5.16, there exists an equation $t \equiv t'$ (or symmetrically, $t' \equiv t$) in $sc$. On one hand, using $\sigma$, we can derive that $\sigma(t) \equiv_{sc} p$ and since $\equiv_{sc}$ is symmetric, $p \equiv_{sc} \sigma(t)$. Also, it follows from reflexivity of $\equiv_{sc}$ that $p' \equiv_{sc} p'$. On the other hand, since $\sigma(t) \xrightarrow{l} p'$ has a proof of depth $n-1$, it follows from the induction hypothesis that $tss \cup \{\textbf{(struct)}\} \vdash \sigma(t) \xrightarrow{l} p'$. Using premises $p \equiv_{sc} \sigma(t)$, $\sigma(t) \xrightarrow{l} p'$ and $p' \equiv_{sc} p'$, we can prove from **(struct)** that $tss \cup \{\textbf{(struct)}\} \vdash p \xrightarrow{l} p'$. ⊠

The above theorem establishes an inclusion result in one direction. To give a full comparison, it remains to give a comparison in the other direction. Next, we give an indirect result leading to such a full comparison.

**Theorem 5.19** Consider a TSS $tss$ in tyft format and structural congruences $sc$. Suppose that bisimilarity with respect to $tss \cup [\![sc]\!]$ is a congruence then the transition relation induced by $tss \cup \{\textbf{(struct)}\}$ is included in the transition relation induced by $tss \cup [\![sc]\!]$ up to bisimilarity.

*Proof.* We have to prove for arbitrary closed terms $p$ and $p'$ and arbitrary label $l$ that if $tss \cup \{\textbf{(struct)}\} \vdash p \xrightarrow{l} p'$ then $tss \cup [\![sc]\!] \vdash p \xrightarrow{l} p''$ and $tss \cup [\![sc]\!] \vdash p'' \underline{\leftrightarrow} p'$. We show this by an induction on the depth of the proof for $p \xrightarrow{l} p'$ in $tss \cup \{\textbf{(struct)}\}$.

If transition $p \xrightarrow{l} p'$ is provable from $tss \cup \{\textbf{(struct)}\}$ with a proof of depth one, then it is due to an axiom in $tss$ and a substitution $\sigma$. By taking the same axiom and substitution, we can prove $tss \cup [\![sc]\!] \vdash p \xrightarrow{l} p'$.

For the induction step, if the transition $p \xrightarrow{l} p'$ is provable from $tss \cup \{\textbf{(struct)}\}$ with a proof of depth $n$, then we distinguish the following two cases.

If the transition is due to a rule in $tss$ of the following form:

$$\textbf{(d)}\frac{\{t_i \xrightarrow{l_i} y_i | i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} t}$$

and a substitution $\sigma$ such that $\sigma(x_i) = p_i$ $(0 \leq i < n)$, $\sigma(t) = p'$ and $p = f(\overrightarrow{p}_{ar(f)-1})$ and take $Y = \{y_i | i \in I\}$. We aim at defining a new substitution $\sigma'$ in a similar way as we have defined it in the proof of Lemma 5.8. Namely, we define that for all $x \in V \setminus Y$, $\sigma'(x) \doteq \sigma(x)$. To complete the definition of $\sigma'$ for the variables in $Y$, we start with the premise of which all variables in the source are defined in $\sigma'$ and the variable in the target is undefined. Such a premise should exist due to well-foundedness of premises with respect to variable-dependency order. Suppose such a premise is $t_i \xrightarrow{l_i} y_i$. Then, since bisimilarity is a congruence and according to the construction of $\sigma'$, it follows from Lemma 4.7 that $\sigma(t_i) \underline{\leftrightarrow} \sigma'(t_i')$. Transition $\sigma(t_i) \xrightarrow{l_i} \sigma(y_i)$ has a proof of depth $n-1$ or less and according to the induction hypothesis, there exists a closed term $p_i'$ such that $\sigma'(t_i) \xrightarrow{l_i} p_i'$ and $\sigma(y_i) \underline{\leftrightarrow} p_i'$. Then, take $\sigma'(y_i) \doteq p_i'$. Using this scheme, we are able to define $\sigma'$ inductively for all variables $y_i$ in such a way that $\forall_{x \in V} \sigma(x) \underline{\leftrightarrow} \sigma'(x)$. This way, we complete a proof for $p \xrightarrow{l} \sigma'(t)$ using $\sigma'$ and **(d)** and it follows from the construction of $\sigma'$ and Lemma 4.7 that $\sigma(t) \underline{\leftrightarrow} \sigma'(t')$.

It remains to prove the case where the transition $p \xrightarrow{l} p'$ is due to **(struct)**:

$$\textbf{(struct)}\frac{x \equiv y \quad y \xrightarrow{l} y' \quad y' \equiv x'}{x \xrightarrow{l} x'}$$

and substitution $\sigma$ such that $\sigma(x) = p$, $\sigma(x') = p'$ and there exists two closed terms $q$ and $q'$ such that $\sigma(y) = q$, $\sigma(y') = q'$, $p \equiv_{sc} q$, $q' \equiv_{sc} p'$ and $q \xrightarrow{l} q'$. Since $q \xrightarrow{l} q'$ has a proof of depth $n-1$, there should exist a closed term $q''$ such that $tss \cup [\![sc]\!] \vdash q \xrightarrow{l} q''$ and $tss \cup [\![sc]\!] \vdash q' \underline{\leftrightarrow} q''$. From $q' \equiv_{sc} p'$ and Lemma 5.17, it follows that $tss \cup [\![sc]\!] \vdash q' \underline{\leftrightarrow} p'$ and since bisimilarity is transitive, it holds that $tss \cup [\![sc]\!] \vdash q'' \underline{\leftrightarrow} p'$ and this concludes the proof. ⊠

The above theorem implies that the three interpretation coincide up to bisimilarity, provided that some necessary conditions (i.e., the tyft format for the TSS and congruence of bisimilarity) hold. Thus, if the necessary conditions of Theorem 5.19 hold, one may choose among these interpretations at will and the result will always be valid for the other interpretations (up to bisimilarity). However, the coincidence result between these interpretations relies on congruence of bisimilarity (w.r.t. the third interpretation) and conformance of the TSS to the tyft format. The following two examples show that when these necessary conditions do not hold, this interpretation may deviate from the first two (and the first interpretation with deduction rule **(struct')** instead of **(struct)**). An explanation of the reason for such necessary conditions comes after each example.

**Example 5.20** Consider the following structural congruence $sc$ and TSS $tss$.

$$a \equiv b \qquad \textbf{(fb)}\frac{}{f(b) \xrightarrow{l_0} b} \qquad \textbf{(b)}\frac{}{b \xrightarrow{l_1} b}$$

If we interpret structural congruences according to our first interpretation, we have that $a \equiv_{sc} b$ and $f(a) \equiv_{sc} f(b)$ and it follows from Lemma 5.4 that $a \underline{\leftrightarrow} b$ and $f(a) \underline{\leftrightarrow} f(b)$. Thus, for example, in this interpretation, transition $f(a) \xrightarrow{l_0} b$ is provable using the above structural congruences and **(struct)**.

According to the third interpretation, $tss \cup [\![sc]\!]$ comprises the following deduction rules:

$$\textbf{(bab)}\frac{a \xrightarrow{l} y}{b \xrightarrow{l} y}(l \in L) \qquad \textbf{(bba)}\frac{b \xrightarrow{l} y}{a \xrightarrow{l} y}(l \in L)$$

$$\textbf{(fb)}\frac{}{f(b) \xrightarrow{l_0} b} \qquad \textbf{(b)}\frac{}{b \xrightarrow{l_1} b}$$

However, from the above TSS, there is no way to prove $f(a) \xrightarrow{l_0} b$, anymore.

One may notice that the above problem has to do with the lack of the congruence property in our TSS; we can derive from Lemma 5.13 that $a \underline{\leftrightarrow} b$ but apparently, it does not hold that $f(a) \underline{\leftrightarrow} f(b)$. This is indeed the case. The deduction rules in $tss$ do not conform to tyft format and even worse, their induced bisimilarity is not a congruence in the first place. Next, we give another counter-example, showing that even if the original TSS is in tyft, adding structural congruences according to the third interpretation may violate the intuition.

**Example 5.21** Consider structural congruences $sc$ and TSS $tss$ as defined below.

$$a \equiv b \qquad f(b) \equiv f(c)$$

$$\textbf{(fx)}\frac{x \xrightarrow{l_0} y}{f(x) \xrightarrow{l_0} f(y)} \qquad \textbf{(c)}\frac{}{c \xrightarrow{l_0} c}$$

Suppose that we have a common signature with constants $a$, $b$ and $c$ and a unary function symbol $f$. Then, according to the first interpretation of structural congruences, we have $a \equiv_{sc} b$ and thus $f(a) \equiv_{sc} f(b)$. Since we also have $f(b) \equiv_{sc} f(c)$, it follows from the transitivity condition that $f(a) \equiv_{sc} f(c)$. According to **(fx)**, $f(c) \xrightarrow{l_0} f(c)$ and thus it follows from **(struct)** that $f(a) \xrightarrow{l_0} f(c)$.

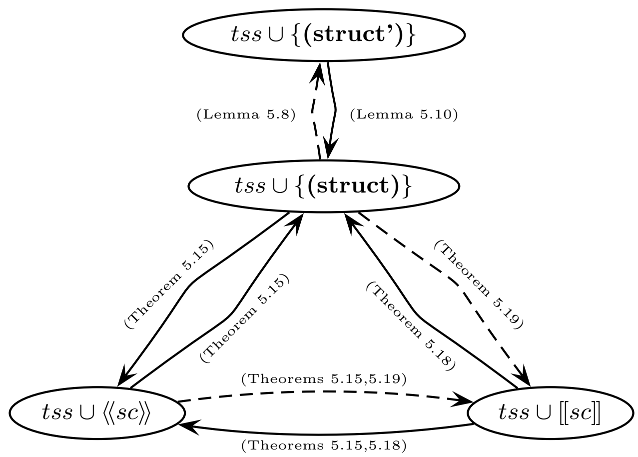


**Figure 5.1** Interpretations of Structural Congruences

According to the third interpretation, $tss \cup [[sc]]$ is defined as:

$$\textbf{(bab)}\frac{a \xrightarrow{l} y}{b \xrightarrow{l} y} \qquad \textbf{(bba)}\frac{b \xrightarrow{l} y}{a \xrightarrow{l} y}$$

$$\textbf{(bfbfc)}\frac{f(b) \xrightarrow{l} y}{f(c) \xrightarrow{l} y} \qquad \textbf{(bfcfb)}\frac{f(c) \xrightarrow{l} y}{f(b) \xrightarrow{l} y}$$

$$\textbf{(fx)}\frac{x \xrightarrow{l_0} y}{f(x) \xrightarrow{l_0} f(y)} \qquad \textbf{(c)}\frac{}{c \xrightarrow{l_0} c}$$

but in the above TSS, transition $f(a) \xrightarrow{l_0} f(c)$ is not provable anymore.

Again the above problem is due to the lack of the congruence property. In the above TSS, it clearly holds that $a \underline{\leftrightarrow} b$ but it does not hold that $f(a) \underline{\leftrightarrow} f(b)$. In the next section, we aim at giving a solution to guarantee this criterion.

Figure 5.1 summarizes the comparison of the three interpretations. In this figure, normal and dashed arrows mean inclusion and inclusion up to bisimilarity of transition relations in the indicated direction, respectively. All the dashed arrows require the tyft format for $tss$ as a necessary condition. For the two cases involving Theorem 5.19, the dashed arrows also require the congruence of bisimilarity for their target interpretation.

Regarding the congruence conditions, note that congruence for the third interpretation implies congruence for the first and the second one, provided that the tyft condition holds (following Corollary 5.7 as the transition relations coincide up to bisimilarity). While congruence of bisimilarity for the first and the second interpretations does not have any general implication for the congruence of bisimilarity with respect to the third one. Recall the following TSS and structural congruences from Example 5.21.

$$a \equiv b \qquad f(b) \equiv f(c)$$

$$\textbf{(fx)}\,\frac{x \xrightarrow{l_0} y}{f(x) \xrightarrow{l_1} f(y)} \qquad \textbf{(c)}\,\frac{}{c \xrightarrow{l_0} c}$$

For the above specification, it can be checked that bisimilarity is a congruence according to the first interpretation (derivable bisimilarities are $a \underline{\leftrightarrow} b$ and $f(a) \underline{\leftrightarrow} f(b) \underline{\leftrightarrow} f(c) \underline{\leftrightarrow} f(f(a)) \underline{\leftrightarrow} \ldots$). However, we have already shown that in the transition relation induced by the third interpretation, bisimilarity is not a congruence as it holds that $a \underline{\leftrightarrow} b$ but not $f(a) \underline{\leftrightarrow} f(b)$. Thus, for our congruence format to be useful for all the three notions, we have to prove it correct with respect to the third interpretation. This way, the congruence format not only induces congruence with respect to the other two notions, it also guarantees that for specifications in the standard format, all the three interpretations coincide and they can be freely chosen at one's convenience.

## 5.4 Congruence for Structural Congruences

In this section, we propose a syntactic format for structural congruences and prove that structural congruences conforming to this format are safe for the purpose of congruence when added to a set of tyft rules. As justified in Section 5.3, we use the third interpretation of structural congruences to prove our format correct. Then, by several counter-examples, we show that none of the syntactic constraints on this format can be dropped in general and thus our syntactic format cannot be relaxed trivially.

### 5.4.1 Congruence Format for Structural Congruences (cfsc)

Our syntactic criteria on structural congruences are defined below.

**Definition 5.22 (Cfsc format)** Structural congruences $sc$ (added to a TSS $tss$) are in the cfsc format if and only if any equation in $sc$ is of one of the following two forms.

1. An $fx$ *equation* is of the form $f(\overrightarrow{x}_{ar(f)-1}) \equiv g(\overrightarrow{y}_{ar(g)-1})$ for function symbols $f$ and $g$ (which need not be different) and for variables $x_i$ $(0 \leq i < ar(f))$ and $y_j$ $(0 \leq j < ar(g))$. Variables $x_i$ (for all $0 \leq i < ar(f)$) and $y_j$ (for all $0 \leq j < ar(g)$) are distinct among themselves (i.e., for all $i \neq j$, $x_i \neq x_j$ and $y_i \neq y_j$) but they need not form two disjoint sets (i.e., it may be that for some $i$ and $j$, $x_i = y_j$).

2. A *defining equation* is of the form $f(x_0, \ldots, x_{ar(f)-1}) \equiv t$ (or similarly, $t \equiv f(\overrightarrow{x}_{ar(f)-1})$ which we do not mention in the remainder due to symmetry) where $f$ is a function symbol and $t$ is an arbitrary term. Similar to $fx$ equations, variables $x_i$ (for all $0 \leq i < ar(f)$) have to be distinct. Two more conditions have to be satisfied for his type of equations; first, all variables in $t$ should be bound by variables $x_0, \ldots, x_{ar(f)-1}$, i.e., $vars(t) \subseteq \{x_i | 0 \leq i < ar(f)\}$ and second, $f$ may not appear in any other structural congruence equation and source of the conclusion of any deduction rule in $tss$. Note that we have no further assumption about $t$, thus, there may be a repetition of variables in $t$, occurrences of $f$ may appear in $t$ and $t$ may consist of any number of constants and function symbols.

The above two categories are not disjoint; i.e., an equation may be both $fx$ and defining. For the remainder, it does not make any difference whether such equations are taken as $fx$, defining, or both.

In the following theorem, we state that structural congruences conforming to the cfsc format induce a congruent bisimilarity relation (with respect to all the three interpretations) when added to a set of tyft rules.

**Theorem 5.23** (Congruence Theorem for cfsc) Consider a set of deduction rules $tss$ in tyft format. If structural congruences $sc$ (added to $tss$) are in the cfsc format, then bisimilarity is a congruence for all the transition relations induced by the three interpretations of $tss$ extended with $sc$.

*Proof.* We prove the theorem for the third interpretation and it follows from Theorem 5.18 and 5.19 and Corollary 5.7 that bisimilarity is a congruence for the first interpretation. Also, from Theorem 5.15, it follows that bisimilarity is congruence for the second interpretation. Since $tss$ is in tyft format, it also follows from Lemmas 5.8 and 5.10 that bisimilarity is a congruence for $tss \cup \{\textbf{(struct')}\}$, as well.

We give an indirect proof for this theorem. First, we give a slightly simplified interpretation of structural congruences in the cfsc format, denoted by $tss \cup [[sc]]^*$. The simplification is only concerned with defining rules. Consider a defining equation of the form $f(x_0, \ldots, x_{ar(f)-1}) \equiv t$; this equation is aimed at defining the operational behavior of $f$, thus the following rule introduced by the third interpretation seems redundant.

$$\textbf{(bft)}\frac{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}{t \xrightarrow{l} y}$$

The simplification only eliminates rules of the above shape. As a consequence of this simplification the resulting TSS is naturally in tyft format. Thus, the congruence of bisimilarity follows from Theorem 3.7. Then, we prove that for a $tss$ in tyft and $sc$ in cfsc, $tss \cup [\![sc]\!]$ and $tss \cup [\![sc]\!]^*$ are equal and we conclude that bisimilarity is a congruence for $tss \cup [\![sc]\!]$, as well.

**Definition 5.24** (Structural Congruences as Bisimilarity: Simplified) Consider structural congruences $sc$ on signature $\Sigma$ and a TSS $tss = (\Sigma, V, L, \{\rightarrow\}, D)$. We define a new TSS $tss \cup [\![sc]\!]^* \doteq (\Sigma, V, L, \{\rightarrow\}, D)$, where $[\![sc]\!]^*$ is the SOS interpretation of $sc$, defined as follows:

$Fx$ equations: $[\![f(\overrightarrow{x}_{ar(f)-1}) \equiv g(\overrightarrow{y}_{ar(g)-1})]\!]^* \doteq$

$$\left\{\begin{array}{l} \textbf{(bfg)}\dfrac{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}{g(\overrightarrow{y}_{ar(g)-1}) \xrightarrow{l} y}(l \in L), \\ \\ \textbf{(bgf)}\dfrac{g(\overrightarrow{y}_{ar(g)-1}) \xrightarrow{l} y}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}(l \in L) \end{array}\right\};$$

Defining equations: $[\![f(\overrightarrow{x}_{ar(f)-1}) \equiv t]\!]^* \doteq$

$$\{\textbf{(fdef)}\frac{t \xrightarrow{l} y}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}(l \in L)\};$$

$[\![sc]\!]^* \doteq \bigcup_{(t \equiv t') \in sc} [\![t \equiv t']\!]^*; s$

where in each of the introduced deduction rules, $y$ is a fresh variable not appearing in the source of any formula in the same deduction rule (i.e., $y \notin \{x_i, y_j | 0 \leq i < ar(f) \wedge 0 \leq j < ar(g)\}$). As stated before, for equations matching both $fx$ and defining equations, one can choose any of the above definitions at will.

It can be easily observed from the above construction that if $tss$ is in the tyft format and $sc$ is in cfsc, then $tss \cup [\![sc]\!]^*$ is in the tyft format. Thus, it follows from Theorem 3.7 that bisimilarity is a congruence for $tss \cup [\![sc]\!]^*$.

Next, we show that for transition systems specification $tss$ in the tyft format and structural congruence $sc$ in the cfsc format, $tss \cup [\![sc]\!]$ and $tss \cup [\![sc]\!]^*$ are equal.

For arbitrary closed terms $p$ and $p'$ and arbitrary label $l$, if $tss \cup [\![sc]\!] \vdash p \xrightarrow{l} p'$, we prove that $tss \cup [\![sc]\!]^* \vdash p \xrightarrow{l} p'$. We use an induction on the depth of the proof for

$p \xrightarrow{l} p'$ in $tss \cup [\![sc]\!]$. The implication in the other direction holds vacuously as the set of deduction rules of $tss \cup [\![sc]\!]^*$ is a subset of that of $tss \cup [\![sc]\!]$.

For the induction basis, the transition has to be due to an axiom in $tss$ and a substitution $\sigma$, thus, using the same axiom and substitution, we can prove the same transition in $tss \cup [\![sc]\!]^*$.

For the induction step, if the transition $p \xrightarrow{l} p'$ in $tss \cup [\![sc]\!]$ is due to a rule that is in $tss \cup [\![sc]\!]^*$, as well, then according to the induction hypothesis, we can prove the premises of this rule from $tss \cup [\![sc]\!]^*$ and since the rule is $tss \cup [\![sc]\!]^*$, we can use the same rule and the same substitution to prove $p \xrightarrow{l} p'$.

It only remains to prove the induction step for the cases where the last rule is not in $tss \cup [\![sc]\!]^*$, thus of the shape:

$$\frac{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}{t \xrightarrow{l} y}$$

corresponding to a defining equation $f(\overrightarrow{x}_{ar(f)-1}) \equiv t$ and there exists a substitution $\sigma$ such that $\sigma(t) = p$ and $\sigma(y) = p'$ and there exist closed terms $p_i$ $(0 \leq i < ar(f))$ such that $\sigma(x_i) = p_i$. The transition $f(\overrightarrow{p}_{ar(f)-1}) \xrightarrow{l} p'$ has a proof of depth $n-1$ and, according to the induction hypothesis, is provable from $tss \cup [\![sc]\!]^*$. Consider the proof of this transition in $tss \cup [\![sc]\!]^*$. Note that since $f(\overrightarrow{x}_{ar(f)-1}) \equiv t$ is a defining equation, $f$ does not appear in the source of the conclusion of any deduction rule in $tss$. Further, since $f$ does not appear in any other equation, there is no rule in $[\![sc]\!]^*$ with $f$ in its source of conclusion, but the following rule.

$$\frac{t \xrightarrow{l} y}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}$$

Thus, the transition $f(\overrightarrow{p}_{ar(f)-1}) \xrightarrow{l} p'$ is due to the above rule and there exists a substitution $\sigma'$ such that $\sigma'(x_i) = p_i$ $(0 \leq i < ar(f))$ and $\sigma'(y) = p'$. But $vars(t) \subseteq \{x_i | 0 \leq i < ar(f)\}$, and $\sigma'(x_i) = \sigma(x_i) = p_i$ $(0 \leq i < ar(f))$ thus, $\sigma'(t) = \sigma(t)$. Hence, $\sigma(t) \xrightarrow{l} p'$ has a proof in $tss \cup [\![sc]\!]^*$.

This concludes the proof, as we have shown that $tss \cup [\![sc]\!]$ is equal to $tss \cup [\![sc]\!]^*$ and $tss \cup [\![sc]\!]^*$ is in the tyft format. ⊠

### 5.4.2 Impossible Relaxations of Cfsc

Next, we show that the cfsc format cannot be relaxed in any obvious way. We take each and every syntactic constraint on cfsc and by an abstract counter-example, show that removing it will result in violating congruence of bisimilarity. The counter-examples will be in such a way that the congruence is ruined according to all three interpretations. We start with a counter-example showing that variables in each side of the $fx$ equation need to be distinct.

**Example 5.25**

$$f(x, x) \equiv a \quad \textbf{(a)}\frac{}{a \xrightarrow{l_0} a} \quad \textbf{(b)}\frac{}{b \xrightarrow{l_0} a}$$

Similar to Example 5.11, it clearly holds in the above specification that $a \underline{\leftrightarrow} b$. However, it does not hold that $f(a, a) \underline{\leftrightarrow} f(a, b)$ since the former can perform an $l_0$ transition, while the latter cannot.

The other condition on $fx$ equations is that they may only have one function symbol in each side of the equation. We have already shown that this constraint cannot be relaxed in Example 5.11 in the previous section. There, the equation $a \equiv f(b)$ had two function symbols, namely the constant $b$ and unary function symbol $f$ and the congruence property is shown to be violated. A similar condition forces defining equations to have only one function symbol on the side to be defined (i.e., only $f$ in the left-hand-side of the equation $f(\overrightarrow{x}_{ar(f)-1}) \equiv t$). In the following example, we show that allowing more function symbols also endangers congruence.

**Example 5.26**

$$f(b) \equiv a \quad \textbf{(a)}\frac{}{a \xrightarrow{l_0} a}$$

Suppose that our signature consists of three constants $a$, $b$ and $c$ and a unary function symbol $f$. Then, it immediately follows that $b \underline{\leftrightarrow} c$ since none of the two constants can perform any transition. However, it does not hold that $f(b) \underline{\leftrightarrow} f(c)$ since the first term can perform a transition while the latter cannot.

The remaining constraints are on defining equations. First of all, for a defining equation $f(x_0, \ldots, x_{ar(f)-1}) \equiv t$, variables $x_i$ should all be distinct. We have already shown in Example 5.25 that relaxing this constraint may be harmful, for the only structural congruence equation satisfies both the definition of $fx$ and defining equations. The other constraint on a defining equation $f(\overrightarrow{x}_{ar(f)-1}) \equiv t$ is that $vars(t) \subseteq \{x_i | 0 \leq i < ar(f)\}$. The following counter-example shows that we cannot drop this constraint.

**Example 5.27**

$$d \equiv f(a, x) \quad \textbf{(c)}\frac{}{c \xrightarrow{l_0} c} \qquad \textbf{(f)}\frac{x_1 \xrightarrow{l_0} y_1}{f(x_0, x_1) \xrightarrow{l_0} y_1}$$

Suppose that our common signature consists of $a$, $b$, $c$ and $d$ as constants and $f$ as a unary function symbol. Equation $d \equiv f(a, x)$ fits all syntactic criteria of a defining equation (for $d$), but the one stated above. It follows from **(f)** that $f(a,c) \xrightarrow{l_0} c$. Since $d \equiv f(a,x)$, then $d \xrightarrow{l_0} c$ and from the same equation (in the other direction), we can deduce that $f(a,b) \xrightarrow{l_0} c$. However, it cannot be derived that $f(b,b) \xrightarrow{l_0} c$. This witnesses that bisimilarity is not a congruence, as $a \underline{\leftrightarrow} b$ but it does not hold that $f(a,b) \underline{\leftrightarrow} f(b,b)$.

The last constraint on defining equations is concerned with freshness of the function symbol being defined. In the following two counter-examples, we show that the defined function symbol cannot appear in any other structural congruence equation, nor in the source of the conclusion of a deduction rule.

**Example 5.28**

$$c \equiv a \quad c \equiv g(b) \quad \textbf{(a)}\frac{}{a \xrightarrow{l_0} a} \quad \textbf{(b)}\frac{}{b \xrightarrow{l_0} a}$$

Again, in the above specification, we have $a \underline{\leftrightarrow} b$ but it is not true that $g(a) \underline{\leftrightarrow} g(b)$ since from the structural congruences, we can derive that $a \equiv_{sc} g(b)$ and hence $g(b)$ can perform an $l_0$ transition to $a$ while $g(a)$ cannot perform any transition.

**Example 5.29**

$$f(x) \equiv g(a) \quad \textbf{(a)}\frac{}{a \xrightarrow{l_0} a} \quad \textbf{(b)}\frac{}{b \xrightarrow{l_0} a} \quad \textbf{(f)}\frac{}{f(x) \xrightarrow{l_0} f(x)}$$

It follows from the above specification that $a \underline{\leftrightarrow} b$ but it does not hold that $g(a) \underline{\leftrightarrow} g(b)$ since the former can perform a transition due to structural congruences and **(f)** while the latter cannot perform any transition.

## 5.5 Negative Premises

As motivated in Section 3.2.4, sometimes it comes handy to define a transition based on the impossibility of a transition for a particular subterm. Several examples (e.g., deadlock detection, sequencing and urgency, cf. [27]) show that negative premises are useful additions to TSS's. Thus, it seems natural to extend TSS's in tyft format to account for negative premises. The ntyft format (Definition 3.8) realizes this goal.

As stated before, in the presence of negative premises, the concepts of proof and provable transitions become more complicated. A proof, as defined before, can provide a reason for presence of a transition but not for its absence. Thus, we have to resort to other notions of proof that can account for absence of transitions, as well. We first consider the notion of supported model (Definition 2.6). Using the interpretations presented in Section 5.3, one can use the notion of supported model for TSS's augmented with structural congruences. However, this may lead to strange phenomena as witnessed by the following example.

**Example 5.30** Consider the following structural congruence equation, added to a TSS with the empty set of rules. Suppose that the common signature comprises of constants $a$ and $b$ and unary function symbols $f$ and $g$.

$$g(x) \equiv f(a)$$

The above equation clearly satisfies the cfsc format as a defining equation and thus bisimilarity is congruence. According to the notion of provable transitions (Definitions 2.5 and 5.3) the above combination of the TSS and structural congruences induces an empty transition relation. However, in addition to this intuitive transition relation, the same combination has another supported model, as well, namely $\{f(a) \xrightarrow{l} a, g(a) \xrightarrow{l} a, g(b) \xrightarrow{l} a, g(f(a)) \xrightarrow{l} a, \ldots\}$ (for which bisimilarity is not a congruence).

The problem in the above example lies in the inherent cyclicity in the structural congruence rule or the corresponding interpretations. For example, in the third interpretation the following two rules are added to the TSS.

$$\frac{f(a) \xrightarrow{l} y}{g(x) \xrightarrow{l} y} \quad \frac{g(x) \xrightarrow{l} y}{f(a) \xrightarrow{l} y}$$

This problem has been observed in the context of pure SOS specifications and led to a number of alternative interpretations or restrictions on TSS's with negative premises. In particular, it is shown that if the TSS is (strictly) stratified (Definition 3.9), it induces a (unique) transition relation (for which bisimilarity is a congruence).

The following theorem from [61] formalizes the advantages of stratified TSS's.

**Theorem 5.31** Consider a TSS *tss* in the ntyft format. If *tss* is stratified, then it has a supported model. If *tss* is strictly stratified, then the supported model is unique. Bisimilarity is a congruence for all supported models of a stratified TSS.

However, as later noted in [27], strict stratification is too much to ask for a unique transition relation. There are several examples of TSS's that intuitively induce a unique transition relation but cannot be strictly stratified. Example 5.30 and any other example in which instances of deduction rules may have a cyclic reference to each other share such a phenomena and were noted in the literature [27, 59]. Hence, we choose the notion of *stable model* (Definition 2.8) that in our mind gives a reasonable and intuitive semantics for TSS with negative premises. The definition is slightly adapted to fit our notation and past definitions.

**Definition 5.32 (Stable Model: Extended)** A positive closed formula $\phi$ is provable from a set of positive formula $T$ and a TSS $tss$, denoted by $(T, tss) \vdash \phi$, if and only if there is an upwardly branching tree of which the nodes are labelled by closed formulae such that

- the root node is labelled by $\phi$, and
- if the label of a node $q$, denoted by $\psi$, is a positive formula and $\{\psi_i \mid i \in I\}$ is the set of labels of the nodes directly above $q$, then there exist a deduction rule $\frac{\{\chi_i \mid i \in I\}}{\chi}$ in $tss$ (where $\chi_i$ can be a negative or a positive formula) and a substitution $\sigma$ such that $\sigma(\chi) = \psi$, and for all $i \in I$, $\sigma(\chi_i) = \psi_i$;
- if the label of a node $q$, denoted by $p \equiv p'$ is a structural congruence, then $p \equiv_{sc} p'$;
- if the label of a node $q$, denoted by $p \overset{l}{\nrightarrow}$, is a negative formula then there exists no $p'$ such that $p \xrightarrow{l} p' \in T$.

A stable model, also called a transition relation, defined by a TSS $tss$ is a set of formulae $T$ such that for all closed positive formulae $\phi$, $\phi \in T$ if and only if $(T, tss) \vdash \phi$.

Note that anomalies, such as those observed in Example 5.30, are resolved in the stable model interpretation. Particularly, the stable model of the TSS in Example 5.30 is now the intuitive empty set. The main reason for this is that the stable model requires a complete proof for positive formulae (as in Definition 2.5) rather than looking for a single matching deduction rule (as in Definition 2.6).

From Theorem 3.10, it follows that bisimilarity with respect to the stable model of a stratified transition systems specification is a congruence.

Now, we have enough ingredients to study the implications of negative premises on the structural congruences. But before doing so, we show that a naive treatment of structural congruences, i.e., neglecting them, may ruin the well-definedness of the induced transition relation.

**Example 5.33**

$$\textbf{(b)}\frac{a \overset{l_0}{\nrightarrow}}{b \xrightarrow{l_0} b}$$

The above TSS (with $a$ and $b$ as constants), is strictly stratified by the function $\mathcal{S}$, if we define for all closed terms $p$, $\mathcal{S}(a \xrightarrow{l_0} p) \doteq 1$ and $\mathcal{S}(b \xrightarrow{l_0} p) \doteq 2$. Following Theorem 3.10, it defines the unique transition relation (its stable model), namely $\{b \xrightarrow{l_0} b\}$.

Suppose that we add the following structural congruence (which is indeed in the cfsc format) to the above TSS:

$$a \equiv b$$

Suddenly, the associated TSS looses its well-definedness. The combination of **(b)** and $a \equiv b$ leads to a contradiction since $b \xrightarrow{l_0} b$ if and only if $a \overset{l_0}{\nrightarrow}$ and if $b \xrightarrow{l_0} b$ then $a \overset{l_0}{\nrightarrow} b$.

To solve the above problem, we extend the notion of stratification to structural congruences as follows.

**Definition 5.34 (Stratification: Extended )** Consider a TSS $tss$ in ntyft format and structural congruence in the cfsc format. We call the combination of $tss$ and $sc$ stratified, if there exists a function $\mathcal{S}$ from closed formulae to an ordinal such that for all closed substitutions $\sigma$:

1. for all rules in $tss$ of the following form:

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} y_i | i \in I\} \quad \{t_j \overset{l_j}{\nrightarrow}_{r_j} | j \in J\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t}$$

   it holds that $\forall_{i \in I} \mathcal{S}(\sigma(t_i \xrightarrow{l_i}_{r_i} y_i)) \leq \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t))$ and $\forall_{j \in J, t' \in \mathcal{T}}$ $\mathcal{S}(\sigma(t_j \xrightarrow{l_j}_{r_j} t')) < \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t))$,

2. for all $fx$ equations of the form $f(\overrightarrow{x}_{ar(f)-1}) \equiv g(\overrightarrow{x}_{ar(g)-1})$ in $sc$, it holds that $\forall_{l \in L, t \in \mathcal{T}}$ $\mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t)) = \mathcal{S}(\sigma(g(\overrightarrow{x}_{ar(g)-1}) \xrightarrow{l}_r t))$,

3. for all defining equations of the form $f(\overrightarrow{x}_{ar(f)-1}) \equiv t$ in $sc$, it holds that $\forall_{l \in L, t' \in \mathcal{T}}$ $\mathcal{S}(\sigma(t \xrightarrow{l}_r t')) \leq \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l}_r t'))$.

The above definition is inspired by the structure of the TSS $tss \cup [\![sc]\!]^*$ (Definition 5.24). In fact, a stratification function for $tss \cup [\![sc]\!]^*$ precisely requires the above conditions to hold.

Next, we extend the well-definedness theorem for the transition relation to the setting with structural congruences. The following theorem states that if a combination of a TSS and structural congruences is stratified, then it defines a unique transition relation.

**Theorem 5.35** If the combination of transition system $tss$ in the ntyft format and structural congruences $sc$ in cfsc is stratified, then $tss \cup [\![sc]\!]$ has a unique stable model.

*Proof.* Consider the TSS $tss \cup [\![sc]\!]^*$, it trivially follows from the hypotheses that it is in ntyft format and stratified. Thus, according to Theorem 3.10, $tss \cup [\![sc]\!]^*$ has a unique stable model. If we also show that the stable models of $tss \cup [\![sc]\!]^*$ and $tss \cup [\![sc]\!]$ coincide, then the thesis follows. This follows from the following claim.

**Claim.** Consider a set of positive closed formulae $T$ (on the common signature $\Sigma$). For all closed terms $p, p' \in \mathcal{C}$ and label $l \in L$ then the following statement holds:

$$(T, tss \cup [\![sc]\!]^*) \vdash p \xrightarrow{l} p' \Leftrightarrow (T, tss \cup [\![sc]\!]) \vdash p \xrightarrow{l} p'$$

*Proof.* We divide this into the following two implications:

1. $(T, tss \cup [\![sc]\!]^*) \vdash p \xrightarrow{l} p' \Rightarrow (T, tss \cup [\![sc]\!]) \vdash p \xrightarrow{l} p'$

   This holds trivially since the deduction rules of $tss \cup [\![sc]\!]^*$ are all in $tss \cup [\![sc]\!]$ and thus the proof for $p \xrightarrow{l} p'$ in $tss \cup [\![sc]\!]^*$ is still valid in $tss \cup [\![sc]\!]$.

2. $(T, tss \cup [\![sc]\!]) \vdash p \xrightarrow{l} p' \Rightarrow (T, tss \cup [\![sc]\!]^*) \vdash p \xrightarrow{l} p'$

   We prove this by an induction on the depth of the proof tree for $(T, tss \cup [\![sc]\!]) \vdash p \xrightarrow{l} p'$.

   For the induction basis, if the proof is of depth 1, then it is due to a rule that is also in $tss \cup [\![sc]\!]^*$ and a substitution $\sigma$ (rules in $[\![sc]\!] \setminus [\![sc]\!]^*$ cannot be used in proof of depth 1 as they have a positive formula in the premise which needs a proof). Using the same rule and the same substitution we can prove this transition from $(T, tss \cup [\![sc]\!])$.

   For the induction step, suppose that the statement holds for closed positive formulae with a proof of depth $n-1$ or less and suppose that $(T, tss \cup [\![sc]\!]) \vdash p \xrightarrow{l} p'$ has a proof of depth $n$. Then, either the last rule is in $tss \cup [\![sc]\!]^*$, as well, from which, using the induction hypothesis on the premises, we can prove that $(T, tss \cup [\![sc]\!]) \vdash p \xrightarrow{l} p'$, or the last rule in the proof structure is in $[\![sc]\!] \setminus [\![sc]\!]^*$. Then, the deduction rule should be of the following form:

$$\frac{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}{t \xrightarrow{l} y}$$

   for a function symbol $f$ and there exists a substitution $\sigma$ such that $\sigma(t) = p$, $\sigma(y) = p'$ and there exists a defining equation $f(\overrightarrow{x}_{ar(f)-1}) \equiv t$ in the structural congruences. Since $(T, tss \cup [\![sc]\!]) \vdash p \xrightarrow{l} p'$, there should exist a deduction rule such that $\sigma(f(\overrightarrow{x}_{ar(f)-1})) \xrightarrow{l} p'$ is provable from $(T, tss \cup [\![sc]\!])$. Since equation $f(\overrightarrow{x}_{ar(f)-1}) \equiv t$ is defining, there is no rule in $tss$ with $f$ appearing in the source of its conclusion (and there is no other equation in $sc$ in which $f$ appears). Thus, the only option for providing a proof for $\sigma(f(\overrightarrow{x}_{ar(f)-1})) \xrightarrow{l} p'$ is a deduction rule of the following shape

$$\frac{t \xrightarrow{l} y}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{l} y}$$

and a substitution $\sigma'$ such that $\sigma'(x_i) = \sigma(x_i)$ (for all $0 \leq i < ar(f)$) and $\sigma(y) = p'$. On one hand, $\sigma'(t) \xrightarrow{l} p'$ is a positive formula, it should have a proof depth less than $n-1$ and thus it follows from the induction hypothesis that $(T, tss \cup [[sc]]^*) \vdash \sigma'(t) \xrightarrow{l} p'$. On the other hand, $vars(t) \subseteq \{x_i | 0 \leq i < ar(f)\}$ and thus, $\sigma'(t) = \sigma(t)$, thus, $(T, tss \cup [[sc]]^*) \vdash \sigma(t) \xrightarrow{l} p'$ and hence $(T, tss \cup [[sc]]^*) \vdash p' \xrightarrow{l} p'$.

⊠

Suppose that $T$ is a stable model of $tss \cup [[sc]]^*$. Then it follows from Definition 5.32 that for all closed formula $\phi$, $\phi \in T$ if and only if $(T, tss \cup [[sc]]^*) \vdash \phi$ and then from the above claim that $\phi \in T$ if and only if $(T, tss \cup [[sc]]) \vdash \phi$. Thus, $T$ is a stable model of $tss \cup [[sc]]$. The reasoning holds in the reverse direction, as well, and thus, the stable models of $tss \cup [[sc]]$ and $tss \cup [[sc]]^*$ coincide.

⊠

We do not intend to extend all the results of Section 5.3 to specifications with negative premises. However, it can be checked that, similar to the above case (for the coincidence of $tss \cup [[sc]]$ and $tss \cup [[sc]]^*$), all the results of Section 5.3 hold for TSS's with negative premises with the additional necessary condition of being stratified. The proofs of the above mentioned results then only need a mere change of notation from $tss \vdash \phi$ to $(T, tss) \vdash \phi$.

Possible extensions to ntyft format are the addition of ntyxt rules and predicates. The ntyft/ntyxt format is a relaxation of ntyft format that allows for variables in the source of the conclusion. In [61], it is shown how to reduce ntyft/ntyxt format to ntyft format. Adding structural congruences to TSS's in the ntyft/ntyxt format, however, is not straightforward. The reduction of ntyft/ntyxt to ntyft requires to copy each ntyxt rule for every function symbol in the signature. This reduction thus disallows the presence of any defining equation, as the new deduction rules contain defined function symbols in the source of their conclusion. Thus, up to now, we can only guarantee congruence for a combination of structural congruences and a TSS with ntyxt rules if the structural congruences comprise of $fx$ equations only. In [97], we suggest to add defining structural congruences to ntyft/ntyxt TSS's as operationally conservative extensions in the sense of Definition 3.12. This way, one can first reduce ntyft/ntyxt TSS's to ntyft ones and then add defining equations to the resulting TSS.

Predicates are other ingredients of TSS's that are used to specify concepts such as termination and divergence on process terms [135]. Unlike negative premises and ntyxt rules, the addition of predicates to a TSS has no implication on structural congruences and the cfsc format. Predicates can be modelled as transitions with a dummy right-hand side (a dummy variable in the premises and a dummy constant in the conclusion). Thus, the results that we have proved so far extend to the PANTH format of [135] which allows for both ntyft-ntyxt rules and predicates. There remains one problem that needs more attention and that is the problem of adding defining equations to TSS's with tyxt rules. This problem is addressed in the next section.

## 5.6 Case Study

In this section, we quote an SOS semantics of CCS from [87] (with restriction to finite sum and introduction of replication operator) and then introduce structural congruences, à la [88], conforming to our format. By doing this, we show how our format is able to capture a number of non-trivial structural congruences and make the presentation look more intuitive and compact. Moreover, from this specification one can still derive congruence for strong bisimilarity automatically.

The syntax of our CCS-like process algebra is given below.

$$P \quad ::= \quad 0 \mid \alpha.P \mid P+Q \mid P \mid\mid Q \mid P \setminus L \mid !P \mid A$$

In this syntax, constant 0 stands for the terminating process. The action prefix operator $\alpha.P$ (which is actually a class of unary operators parameterized by labels $\alpha \in \mathcal{L}$) shows $\alpha$ as its first step and proceeds with $P$. The set of labels $\mathcal{L}$ is partitioned into the set of names, typically denoted by $l$, and co-names, denoted by $\bar{l}$. By extending the same notation, let $\bar{\bar{l}}$ be defined as $l$. Restriction operator $P \setminus L$, parameterized by $L \subseteq \mathcal{L}$ defines the scope of local names (and co-names). Nondeterministic choice is denoted by $+$. Parallel composition is denoted by $P \mid\mid Q$. Parallel replication of process $P$ is denoted by $!P$ which usually serves as a restricted substitute for recursion. Recursive symbols $A$ serve as abbreviations for their defining processes, denoted by $A \doteq P$ and are used to define processes hierarchically. We treat recursive symbols as constants in our signature.

The TSS defining the semantics of our language is given in Figure 5.2. In this specification, rule **(Act)** defines that an action prefix operator can execute its first action and continue with the rest. Each rule in this specification should be considered as a rule schema, representing a possibly infinite number of rules for each $l \in \mathcal{L}$. Side conditions, in this particular case study, only govern presence

$$\textbf{(Act)}\,\frac{}{\alpha.x \xrightarrow{\alpha} x} \qquad \textbf{(Res)}\,\frac{x \xrightarrow{\alpha} y}{x \setminus L \xrightarrow{\alpha} y \setminus L}\,(\alpha, \overline{\alpha} \notin L)$$

$$\textbf{(Sum0)}\,\frac{x_0 \xrightarrow{\alpha} y}{x_0 + x_1 \xrightarrow{\alpha} y} \qquad \textbf{(Sum1)}\,\frac{x_1 \xrightarrow{\alpha} y}{x_0 + x_1 \xrightarrow{\alpha} y}$$

$$\textbf{(Com0)}\,\frac{x_0 \xrightarrow{\alpha} y_0}{x_0 \mid\mid x_1 \xrightarrow{\alpha} y_0 \mid\mid x_1} \qquad \textbf{(Com1)}\,\frac{x_1 \xrightarrow{\alpha} y_1}{x_0 \mid\mid x_1 \xrightarrow{\alpha} x_0 \mid\mid y_1}$$

$$\textbf{(Com2)}\,\frac{x_0 \xrightarrow{l} y_0 \quad x_1 \xrightarrow{\bar{l}} y_1}{x_0 \mid\mid x_1 \xrightarrow{\tau} y_0 \mid\mid y_1}$$

$$\textbf{(Con)}\,\frac{t \xrightarrow{\alpha} y}{A \xrightarrow{\alpha} y}\,(A \doteq t) \qquad \textbf{(Rep)}\,\frac{x \mid\mid !x \xrightarrow{\alpha} y}{!x \xrightarrow{\alpha} y}$$

$$l, \bar{l} \in \mathcal{L}, \alpha \in \mathcal{L} \cup \{\tau\}$$

**Figure 5.2** Semantics of CCS: SOS rules

and absence of such copies. Rule **(Res)** allows for performing actions beyond the restricted set $L$ (i.e., blocks the rest). Rules **(Sum0)** and **(Sum1)** define the nondeterministic choice operator. Rules **(Com0)** and **(Com1)** define the interleaving behavior of parallel composition and rule **(Com2)** defines its communication (synchronization) behavior. A particular label $\tau$ is added for inactions resulting from communication and $\overline{\tau}$ is defined as $\tau$. Rule **(Con)** shows how recursive constants represent the behavior of their defining terms and finally, **(Rep)** defines the concept of replication.

By using our format, we can copy a number of structural congruences, defined in [88] for the $\pi$-calculus and thus, eliminate some of the deduction rules. The result is shown in Figure 5.3.

Note that all of the SOS rules are in tyft format and the top two structural congruence equations are $fx$ equations while the bottom ones are defining equations. Thus, one may easily deduce from Theorem 5.23 that strong bisimilarity is a congruence with respect to the induced transition relation. This can already be considered an achievement. However, one may argue that we could not specify

$$(\textbf{Act})\,\frac{}{\alpha.x \xrightarrow{\alpha} x} \qquad (\textbf{Res})\,\frac{x \xrightarrow{\alpha} y}{x \setminus L \xrightarrow{\alpha} y \setminus L}\,(\alpha, \overline{\alpha} \notin L)$$

$$(\textbf{NSum0})\,\frac{x_0 \xrightarrow{\alpha} y}{x_0 + x_1 \xrightarrow{\alpha} y}$$

$$(\textbf{NCom0})\,\frac{x_0 \xrightarrow{\alpha} y_0}{x_0 \mid\mid x_1 \xrightarrow{\alpha} y_0 \mid\mid x_1} \qquad (\textbf{NCom1})\,\frac{x_0 \xrightarrow{l} y_0 \quad x_1 \xrightarrow{\bar{l}} y_1}{x_0 \mid\mid x_1 \xrightarrow{\tau} y_0 \mid\mid y_1}$$

$$(\textbf{struct})\,\frac{x \equiv y \quad y \xrightarrow{l} y' \quad y' \equiv x'}{x \xrightarrow{l} x'}\,(l \in L) \qquad \begin{array}{c} x_0 + x_1 \equiv x_1 + x_0 \\ A \equiv t \quad (A \doteq t) \end{array} \qquad \begin{array}{c} x_0 \mid\mid x_1 \equiv x_1 \mid\mid x_0 \\ !x \equiv x \mid\mid !x \end{array}$$

$$l, \bar{l} \in \mathcal{L}, \alpha \in \mathcal{L} \cup \{\tau\}$$

**Figure 5.3** Semantics of CCS: SOS rules with Structural Congruences

some, may be more interesting, structural congruences of [88] such as those for associativity (for parallel composition and nondeterministic choice), idempotency (for nondeterministic choice) and zero element (again for both parallel composition and choice). Our answer to this criticism is that first, in this particular case, all of these properties can be proven from the above specification as theorems and second, there are cases where the very same structural congruences (i.e, associativity, idempotency and zero element) can be harmful for congruence. Next, we give an intuitive example of an associativity equation that harms the congruence property.

**Example 5.36** Take the semantics of our CCS-like language defined before. Suppose that we extend our syntax and semantics with a binary operator $\bullet$. The deduction rule for this operator is given below (note that the deduction rule conforms to the tyft format):

$$(\textbf{LMer})\,\frac{x_0 \xrightarrow{\alpha} y_0}{x_0 \bullet x_1 \xrightarrow{\alpha} y_0 \mid\mid x_1}$$

According to the above rule, this operator forces the first action to be taken by the left-hand-side argument and then turns into a normal parallel composition operator. (Up to here, this operator is similar to the left-merge operator $\|\!\!\_$ of [13] which is usually used for finite axiomatization of parallel composition.) This operator, as defined by rule **(LMer)** is not associative. But, suppose that we also add the following equation to our set of structural congruences, to make it associative.

$$x_0 \bullet (x_1 \bullet x_2) \equiv (x_0 \bullet x_1) \bullet x_2$$

Then, we can easily observe that the congruence property is ruined. For example, it holds that $0 \underline{\leftrightarrow} 0 \bullet \alpha$ (where $\alpha$ is a shorthand for $\alpha.0$), since none of the two can perform any action. However, it does not hold that $\alpha \bullet 0 \ \underline{\leftrightarrow} \ \alpha \bullet (0 \bullet \alpha)$. The left-hand term can only perform an $\alpha$ action and terminate (the structural congruence rule cannot help this term perform more actions since it should contain at least two left-merge operators to fit the structure of the rule). While the right-hand-term is congruent to $(\alpha \bullet 0) \bullet \alpha$ and this new term can perform two consecutive $\alpha$ actions after the first of which it turns into $(0 \mid\mid 0) \mid\mid \alpha$.

## 5.7 Conclusions

In this chapter, we presented a number of ways to interpret structural congruences inside the transition system specification (TSS) framework and compared the outcomes formally. We also defined a syntactic format for structural congruences that makes them safe with respect to the congruence of strong bisimilarity, once they are used in combination with a set of standard (e.g., tyft) SOS rules. To allow for negative premises in the TSS's, the relationship between negative premises in the deduction rules, structural congruences and well-definedness of the transition relation was investigated and sufficient well-definedness criteria were established. To show the application of our format to a concrete example, we applied our syntactic format to a CCS-like process algebra.

Extending the syntactic format to other notions of equivalence and refinement is a possible extension of our work (following the approach of other standard formats for weaker notions of bisimulation, e.g., RBB format of [23]). Studying structural congruences in the bi-algebraic framework of [123] may lead to a foundational framework for this mixed setting, as well. Incorporating the concepts of names and binders in our framework allows us to deal with more interesting instances of process calculi in which structural congruences play an essential role (e.g., [88, 89]). We consider this as an important future extension to our framework.

# Chapter 6

# Conservativity

"*Conservative, n.* A statesman who is enamored of existing evils, as distinguished from the Liberal, who wishes to replace them with others."

["Devil's Dictionary", Ambrose Bierce]

A summarized version of this chapter has appeared as: M.R. Mousavi, M.A. Reniers, Orthogonal Extensions in Structural Operational Semantics, In L. Caires, G.P. Italiano, L. Monteiro, C. Palamidessi and M. Yung eds., *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, Lisbon, Portugal, volume 3580 of Lecture Notes in Computer Science, pp. 1214-1225, Springer-Verlag, July 2005.

## 6.1 Introduction

Programming languages and process calculi have been subject to constant extensions. It is often crucial to make sure that such extensions do not change the intuition behind the old subset, or said otherwise, the extensions are *conservative*. In the context of languages with Structural Operational Semantics (SOS) [108], this topic has been touched upon in [61, 64] and studied in depth in [5, 11, 48, 84, 134, 4]. This research has resulted in meta-theorems proving sufficient conditions for an extension to be *operationally* and/or *equationally conservative*. In the remainder, we mostly refer to [48] which gives the most detailed account of the problem and subsumes almost all previous results. We do not treat multi-sorted and variable binding signatures, addressed in [48, 84], in this chapter.

So far, *operational conservativity* has only allowed for extensions that consistently deny the addition of any new behavior to the old syntax. One can imagine that an extension which grants a new behavior consistently to the old syntax can also be considered safe or “conservative”. This phenomenon occurs quite often in practice. For example, designers of many timed extensions of existing formalisms (e.g., the timed process algebras of [14, 8, 76, 133]) have decided to add timed behavior homogenously to the terms from the old syntax. Unfortunately, it turns out that the existing definitions and their corresponding meta-theorems come short of any formal result about such extensions.

In this chapter, we present a more liberal notion of operational conservativity, called *orthogonality*, which caters for both possibilities (i.e., denying some types of behavior from the old syntax while granting some other types). We show that our notion is useful in the aforementioned cases where the old notions cannot be used. We formulate orthogonality meta-theorems for languages with Structural Operational Semantics and prove them correct.

In [134], *equational extensions* are considered in the setting where a new set of axioms is added to an existing set. Then, the extension is called *equationally conservative* if it induces exactly the same derivable ground (i.e., closed) equalities on the old syntax as the original equational theory. In this chapter, we remove the requirement for including the old set of axioms in the extended equational theory. We refer to such extensions as *equationally conservative ground-extensions*. This relaxation is motivated by the fact that in many extensions, such as those of [14, 109, 133], for some axioms, only all closed derivable equalities on the old syntax are kept and the axioms themselves are removed. This may be due to two reasons: the old axioms do not hold with respect to the newly introduced operators or they (or all their closed instantiations) are derivable from the new axioms. For example, some of the old axioms of [14, 109, 133] (and also Section 6.2 of this chapter) are not sound in the extended language when they are instantiated by terms from the extended syntax. For the relaxed notion of equational conservativity, we present similar meta-theorems as for the old notions in [5, 134, 4]. Operational conservativity is usually considered as a means for equational conservativity and we show that our notion of orthogonality leads to equational conservativity in the same way as operational conservativity does (no matter which notion of equational conservativity is chosen, the traditional notion or the relaxed one).

The rest of this chapter is structured as follows. Section 6.2 presents our relaxed notion of equational conservativity. Orthogonality and related notions are defined in Section 6.3. Subsequently, Section 6.4 defines sufficient conditions for orthogonality. In the same section, we also present theorems establishing the link between orthogonality and equational conservativity. Finally, Section 6.5 summarizes the results and presents future directions. In each section, we provide abstract and concrete examples from the area of process algebra to motivate the definitions and illustrate the results. In this chapter, we recall the definitions of TSS's with constant labels (Definition 3.1 with a single transition relation), stratification (Definition 3.9), operational conservativity and its meta-theorem (Definition 3.12 and Theorem 3.15) and equational theories (Definition 3.16) without re-stating them.

## 6.2 Equational Conservativity

In Definitions 3.12 and 3.16, we defined the notions of operational conservativity and equational theory. In process algebraic formalisms, the notion of equational theory is central and operational conservativity is a means to ensure equational conservativity, as defined below.

**Definition 6.1 (Equational Conservativity)** An equational theory $(\Sigma_1, V, E_1)$ is an *equationally conservative ground-extension* of $(\Sigma_0, V, E_0)$ when $\Sigma_0 \subseteq \Sigma_1$ and for all $p, p' \in \mathcal{C}(\Sigma_0)$, $E_0 \vdash p = p' \Leftrightarrow E_1 \vdash p = p'$.

It is worth mentioning that the above definition is more liberal than the notion of equational conservativity in [134] in that there, it is required that the same axioms are included in the extended equational theory (i.e., $E_0 \subseteq E_1$). In practice, some process algebras do not keep the same axioms when extending the formalism while they make sure that the ground instantiations of the old axioms with old terms indeed remain derivable (see for example, [14, 109, 133] and Example 6.6 in the remainder). Hence, we believe that the restriction imposed by [134] unnecessarily limits the applicability of the theory. If, for any reason, one chooses the more restricted notion of [134], the theorems concerning equational conservativity in this chapter remain valid.

To have a better idea of the concepts introduced so far, we define a Minimal Process Algebra (with its equational and operational theories) and extend it to the timed settings. We study the relationship between the MPA and its timed extension throughout this chapter.

**Example 6.2** (*MPA*: Operational Semantics) Consider the following deduction rules defined on a signature with a constant $\delta$, a family of unary operators $a.\_$ (for all $a \in A$, where $A$ is a given set of atomic actions) and a binary operator $\_ + \_$. The labels of transitions are $a \in A$.

$$\textbf{(a)}\,\frac{}{a.x \xrightarrow{a} x}(a \in A) \quad \textbf{(c0)}\,\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \quad \textbf{(c1)}\,\frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

This TSS (called $tss_m$ in the remainder) is supposed to define a transition relation for the Minimal Process Algebra (*MPA*) of [14], simplified here by removing the concept of termination, which we use as our running example in the remainder. Deduction rules of *MPA* are (strictly) stratified using a measure of size on the terms in the source of formulae and it defines a unique transition relation by all possible interpretations. The following transitions are among those included in this relation: $tss_m \vDash (a.\delta) + \delta \xrightarrow{a} \delta$ and $tss_m \vDash a.(\delta + a.\delta) \xrightarrow{a} \delta + a.\delta$ which are all provable in $tss_m$ using an empty set of negative premises.

**Example 6.3** (*MPA*: Equational Theory) Consider the Minimal Process Algebra of Example 6.2. The following is an axiomatization of *MPA* [14].

$$x + y = y + x \qquad x + (y + z) = (x + y) + z \qquad x + x = x \qquad x + \delta = x$$

It is well-known that this axiomatization is sound and ground-complete with respect to $tss_m$ given in Example 6.2 and strong bisimilarity as the notion of behavioral equivalence (see, for example, [87]). The following are examples of derivable equalities from the above axiomatization: $(a.\delta) + \delta = a.\delta$ and $(a.\delta) + a.\delta = a.\delta$.

Next, we extend the MPA with an aspect of timing.

**Example 6.4** (Timed-*MPA*: Operational Semantics) Consider the following deduction rules (divided into three parts) which are defined on a signature with two constants $\delta$ and $\underline{\underline{\delta}}$, a unary function symbol $\underline{\underline{\sigma}}.\_$, two families of unary function symbols $a.\_$ and $\underline{\underline{a}}.\_$ (for all $a \in A$) and a binary function symbol $\_ + \_$. The set of labels of the TSS is $A \cup \{1\}$ (with $1 \notin A$).

(1) $$\textbf{(ua)}\,\frac{}{\underline{\underline{a}}.x \xrightarrow{a} x} \quad \textbf{(td)}\,\frac{}{\underline{\underline{\sigma}}.x \xrightarrow{1} x}$$

(2) $$\textbf{(tc0)}\,\frac{x \xrightarrow{1} x' \quad y \xrightarrow{1} y'}{x + y \xrightarrow{1} x' + y'} \quad \textbf{(tc1)}\,\frac{x \xrightarrow{1} x' \quad y \stackrel{1}{\nrightarrow}}{x + y \xrightarrow{1} x'} \quad \textbf{(tc2)}\,\frac{y \xrightarrow{1} y' \quad x \stackrel{1}{\nrightarrow}}{x + y \xrightarrow{1} y'}$$

(3) $$\textbf{(ta)}\,\frac{}{a.x \xrightarrow{1} a.x} \quad \textbf{(d)}\,\frac{}{\delta \xrightarrow{1} \delta}$$

The above TSS, which we call $tss_t$ defines the aspect of timing in terms of new time-transitions $\xrightarrow{1}$ and it is added in [14] to $tss_m$ in Example 6.2 to define a relative-discrete-time extension of *MPA*. The intuition behind the new underlined function symbols ($\underline{\underline{a}}.\_$ and $\underline{\underline{\sigma}}.\_$) is that they are not delayable in time and should take their (respectively action and time) transitions immediately. Addition of the first and/or the second parts of the above TSS (each or both) to $tss_m$ results in an operationally conservative extension of the latter as the newly added transitions will be restricted to the new syntax. (Note that in the first and second parts, there is no rule about timed transition of constants in old syntax.) We prove this claim formally as an instance of a meta-theorem in the rest of this chapter. However, the addition of part (3) violates the operational conservativity of the extension as it adds time-transitions ($\xrightarrow{1}$) to the behavior of terms from the old syntax. For example, in combination with part (2), it allows for transitions such as $tss_m \cup tss_t \vDash a.\delta \xrightarrow{1} a.\delta$ and $tss_m \cup tss_t \vDash (a.\delta) + \delta \xrightarrow{1} (a.\delta) + \delta$, all of which are prohibited by the original TSS and thus are considered harmful from the operational conservativity point of view.

As it turns out, the notion of operational conservativity (Definition 3.12) is too restrictive to capture extensions of the above sort. This is illustrated in the following example.

**Example 6.5** (Timed-*MPA*: Operational Conservativity, Revisited) The addition of parts (1) and (2) of $tss_t$ in Example 6.4 to the $tss_m$ of Example 6.2 results in an operationally conservative extension following Theorem 3.15: All three deduction rules of $tss_m$ are source dependent; Rules **(ua)** and **(td)** both have a new function symbol in the source of their conclusion (i.e., $\underline{\underline{a}}.\_$ and $\underline{\underline{\sigma}}.\_$, respectively) and hence, satisfy condition 2(a); Rules **(tc0)**, **(tc1)** and **(tc2)** all have a source-dependent positive premise with a timed-*MPA* label (1) and hence, satisfy condition 2(b). Note that the reduced version of each of the deduction rules **(tc0)**, **(tc1)** and **(tc2)** is that deduction rule itself.

Also, the traditional notion of equational conservativity cannot capture the extension of the following equational theory of timed-*MPA*.

**Example 6.6** (Timed-*MPA*: Equational Theory) Consider the TSS resulting from extending $tss_m$ of Example 6.2 with $tss_t$ of Example 6.4. The following are a set of sound and ground-complete axioms (w.r.t. strong bisimilarity) for this TSS:

$$(1)\ x + y = y + x \qquad (2)\ x + (y + z) = (x + y) + z \qquad (3)\ x + x = x$$

$$(4)\ \delta = \underline{\underline{\sigma}}.\delta \qquad (5)\ x + \underline{\underline{\delta}} = x$$

$$(6)\ (\underline{\underline{\sigma}}.x) + \underline{\underline{\sigma}}.y = \underline{\underline{\sigma}}.(x + y) \qquad (7)\ a.x = (\underline{\underline{a}}.x) + \underline{\underline{\sigma}}.a.x \qquad (8)\ (a.x) + \delta = a.x$$

The above axiomatization underscores the fact we mentioned before. Namely, the axioms of the old system do not hold in the new system (e.g., $(\underline{\underline{a}}.x) + \delta \neq \underline{\underline{a}}.x$ as an instance of $x + \delta = x$) but all closed instantiations of the old axioms by terms of the old syntax are derivable from the new set of axioms.

It can be checked that the above axiomatization of timed-*MPA* is indeed an equationally conservative ground-extension of the axiomatization of *MPA* in the sense of Definition 6.1. Thus, if one considers operational conservativity as a means to equational conservativity, this example already suggests the need for an extension of Definition 3.12. In other words, we believe that the transitions added by the extension are quite innocent and harmless to the intuition behind the original semantics, for they are added uniformly to the old syntax without changing the old behavior or violating previously valid equalities. In the next section, we formalize our idea of orthogonal extensions which caters for extensions of the above type.

## 6.3 Orthogonality

### 6.3.1 Orthogonal Extension

In this section, we define the notion of orthogonality and an instance of this notion, called *granting extensions*, which can be checked syntactically.

**Definition 6.7 (Orthogonal Extension)** Consider TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$ and a behavioral notion of equality $\sim$. The TSS $tss_0 \cup tss_1$ is a $\sim$-*orthogonal* extension of $tss_0$ when

1. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)}\ \forall_{l \in L_0}\ tss_0 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_0 \cup tss_1 \vDash p \xrightarrow{l} p'$, and
2. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)}\ tss_0 \vDash p \sim p' \Leftrightarrow tss_0 \cup tss_1 \vDash p \sim p'$.

Our results in this chapter are valid for most notions of behavioral equivalence in the literature (to be named explicitly in the remainder). The notion of *operational conservativity up to $\phi$-equivalence* of [134, 11] can be seen as a variant of orthogonality which only has the second condition. To our knowledge, beyond operational conservativity results (e.g., [134]), no systematic study of these notions (i.e., orthogonality and operational conservativity up-to, including meta-theorems guaranteeing them) has been carried out.

The following corollary is a direct result of Definition 6.7.

**Corollary 6.8** An operationally conservative extension is an orthogonal extension.

### 6.3.2 Granting Extension

Corollary 6.8 addresses operational conservativity as an extreme case of orthogonality which denies all new transitions from the old syntax; the other extreme is an extension which grants all new behavior to the old syntax. However, for such an extension to be orthogonal, these transitions should be made to equivalent terms from the old syntax. In particular, if we allow for self-transitions, we are able to prove orthogonality with respect to many notions of behavioral equivalence. The following definitions and the subsequent theorem substantiate these concepts.

**Definition 6.9 (Granting Extension)** Consider TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$ with disjoint labels. We call $tss_0 \cup tss_1$ a *granting extension* of $tss_0$ when

1. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)}\ \forall_{l \in L_0}\ tss_0 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_0 \cup tss_1 \vDash p \xrightarrow{l} p'$, and
2. $\forall_{p \in \mathcal{C}(\Sigma_0)}\ \forall_{p' \in \mathcal{C}(\Sigma_0 \cup \Sigma_1)}\ \forall_{l \in L_1}\ tss_0 \cup tss_1 \vDash p \xrightarrow{l} p' \Leftrightarrow p = p'$.

The above definition states that granting extensions keep the old transitions on the old terms intact and only add self-transitions with all of the new labels to old terms. The above definition does not make any statement about the transitions on the new terms, i.e., terms from $\mathcal{C}(\Sigma_0 \cup \Sigma_1) \setminus \mathcal{C}(\Sigma_0)$. We are doubtful whether a meaningful relaxation of Definition 6.9 is possible that allows for anything coarser than syntactic equality on the old terms involved in (the left- or the right-hand side of) the new transitions and still can be captured by simple syntactic checks (which is our aim in this chapter, even if one confines oneself to $\sim$-orthogonality for a particular $\sim$). This suggests that to formulate syntactic criteria for proving orthogonality, we have to resort to one of the two extremes (operational conservativity or granting extensions). Admitting that these two extremes need to be combined in some way to reach a reasonable balance, we define sufficient criteria for this mixture to be orthogonal in the next section. Next, we show that granting extensions are indeed $\sim$-orthogonal extensions for most notions of behavioral equivalence $\sim$.

**Theorem 6.10** Consider TSS's $tss_0$ and $tss_1$ where $tss_1$ is a granting extension of $tss_0$. Let $\sim$ be any of the following notions of behavioral equivalence (cf. [56, 57] and the proof presented next, for details about these notions):

1. trace equivalence $=_{\mathrm{T}}$,
2. failures equivalence $=_{\mathrm{F}}$,
3. ready equivalence $=_{\mathrm{R}}$,
4. failure trace equivalence $=_{\mathrm{FT}}$,

5. ready trace equivalence $=_{\mathrm{RT}}$,

6. simulation equivalence $\rightleftarrows$,

7. ready simulation equivalence $=_{\mathrm{RS}}$,

8. weak bisimilarity $\underline{\leftrightarrow}_{\mathrm{w}}$,

9. branching bisimilarity $\underline{\leftrightarrow}_{\mathrm{b}}$,

10. rooted branching bisimilarity $\underline{\leftrightarrow}_{\mathrm{rb}}$,

11. rooted weak bisimilarity $\underline{\leftrightarrow}_{\mathrm{rw}}$,

12. bisimulation equivalence $\underline{\leftrightarrow}$ ,

then $tss_1$ is a $\sim$-orthogonal extension of $tss_0$.

*Proof.* Let $tss_0 \doteq (\Sigma_0, V, L_0, D_0)$ and $tss_1 \doteq (\Sigma_1, V, L_1, D_1)$ and let $L' \doteq L_1 \setminus L_0$. Copying Definition 6.9, we have:

1. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)}\ \forall_{l \in L_0}\ tss_0 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_1 \vDash p \xrightarrow{l} p'$, and

2. $\forall_{p \in \mathcal{C}(\Sigma_0)}\ \forall_{p' \in \mathcal{C}(\Sigma_1)}\ \forall_{l' \in L'}\ tss_1 \vDash p \xrightarrow{l'} p' \Leftrightarrow p = p'$.

The first item above is the same as the first item in Definition 6.7 of orthogonality. Hence, we have to prove the statement

$$\forall_{p,p' \in \mathcal{C}(\Sigma_0)} tss_0 \vDash p \sim p' \Leftrightarrow tss_1 \vDash p \sim p'$$

for all of the following notions of behavioral equivalence $\sim$ mentioned in the theorem.

1. trace equivalence $=_{\mathrm{T}}$: We start with the following auxiliary definitions

   **Definition 6.11** Let $L^*$ be *the set of all traces* that can be generated from labels $L$ (including the empty trace $\epsilon$). Given a trace $\sigma \in L^*$ and a set of labels $L'$ *the granting extension of* $\sigma$ *with* $L'$, denoted by $\sigma \uparrow L'$, is the smallest set of traces that satisfies $\forall_{\sigma_0, \sigma_1 \in (L \cup L')^*}$ and $\forall_{l \in L'}$:

   (a) $\sigma \in \sigma \uparrow L'$;

   (b) $\sigma_0 \sigma_1 \in \sigma \uparrow L' \Rightarrow \sigma_0 (l) \sigma_1 \in \sigma \uparrow L'$.

where juxtaposition denotes concatenation. For a set of traces $TR$, $TR \uparrow L' \doteq \bigcup_{\sigma \in TR} \sigma \uparrow L'$. Similarly, for a trace $\sigma$ on labels $L$, the trace $\sigma \downarrow L'$ is defined inductively by:

$$\epsilon \downarrow L' \doteq \epsilon, \qquad (l)\sigma \downarrow L' \doteq \begin{cases} \sigma \downarrow L' & l \in L', \\ (l)(\sigma \downarrow L') & l \notin L'. \end{cases}$$

For a set of traces $TR$, $TR \downarrow L' \doteq \{\sigma \downarrow L' \mid \sigma \in TR\}$.

**Corollary 6.12** Consider sets of traces $TR$ and $TR'$ both defined on a set of labels $L$.

(a) If $TR = TR'$ then for an arbitrary $L'$, $TR \downarrow L' = TR' \downarrow L'$ and $TR \uparrow L' = TR' \uparrow L'$;

(b) For a set $L'$ disjoint from $L$, $(TR \uparrow L') \downarrow L' = TR$.

**Lemma 6.13** Consider sets of traces $TR$ and $TR'$ both defined on a set of labels $L$ and a set $L'$ disjoint from $L$. $TR = TR'$ if and only if $TR \uparrow L' = TR' \uparrow L'$.

*Proof.* The implication from left to right follows trivially from the definition of $TR \uparrow L'$ (see the first item of Corollary 6.12). Then, it remains to prove $TR = TR'$ assuming the left-to-right implication and $TR \uparrow L' = TR' \uparrow L'$. It follows from the first item of Corollary 6.12 that $(TR \uparrow L') \downarrow L' = (TR' \uparrow L') \downarrow L'$ and from the second item of the same corollary, $TR = TR'$. ⊠

**Definition 6.14** Given a transition relation $\rightarrow \subseteq \mathcal{C} \times L \times \mathcal{C}$, *the reflexive and transitive closure of* $\rightarrow$, denoted by $\rightarrow^* \subseteq \mathcal{C} \times L^* \times \mathcal{C}$ is defined as the smallest relation satisfying the following constraints: $\forall_{p,p',p'' \in \mathcal{C}}$

(a) $p \xrightarrow{\epsilon}{}^* p$;

(b) $p \xrightarrow{l} p' \Rightarrow p \xrightarrow{(l)}{}^* p'$;

(c) $p \xrightarrow{l} p' \wedge p' \xrightarrow{\sigma}{}^* p'' \Rightarrow p \xrightarrow{(l)\sigma}{}^* p''$.

Let $tss = (\Sigma, V, L, D)$ be a TSS. *The set of traces in tss originating from* $p \in \mathcal{C}$, denoted by $TR(tss, p)$ is the smallest set satisfying for all $p' \in \mathcal{C}$ and $\sigma \in L^*$: if $tss \vDash p \xrightarrow{\sigma}{}^* p'$ then $\sigma \in TR(tss, p)$. The processes $p$ and $q$ are trace equivalent w.r.t. TSS $tss$, notation $tss \vDash p =_{\mathrm{T}} q$, iff $TR(tss, p) = TR(tss, q)$.

**Corollary 6.15** Let $tss_0$ and $tss_1$ be the TSS's defined above. For all $p \in \mathcal{C}(\Sigma_0)$, $TR(tss_1, p) = TR(tss_0, p) \uparrow L'$.

We now have $tss_0 \vDash p =_{\mathrm{T}} q$ iff, by definition, $TR(tss_0, p) = TR(tss_0, q)$ iff, by Lemma 6.13, $TR(tss_0, p) \uparrow L' = TR(tss_0, p) \uparrow L'$ iff, by Corollary 6.15, $TR(tss_1, p) = TR(tss_1, q)$ iff, by definition, $tss_1 \vDash p =_{\mathrm{T}} q$.

2. Failures equivalence $=_{\mathrm{F}}$:

**Definition 6.16** Let $tss = (\Sigma, V, L, D)$ be a TSS. *A pair* $(\sigma, X) \in L^* \times \mathbb{P}(L)$ *is a failure pair of* $p \in \mathcal{C}$ *originating from* $tss$ if $tss \vDash p \xrightarrow{\sigma}{}^* p'$ for some $p'$ such that for all $l \in X$, $tss_0 \vDash p' \overset{l}{\nrightarrow}$. The set of failure pairs in $tss$ originating from $p \in \mathcal{C}$, denoted by $FP(tss, p)$ is the set containing all failure pairs of $p$ originating from $tss$. The processes $p$ and $q$ are failures equivalent w.r.t. TSS $tss$, notation $tss \vDash p =_{\mathrm{F}} q$, iff $FP(tss, p) = FP(tss, q)$.

For a set of failure pairs $FP$ and a set of labels $L$, we define $FP \downarrow L \doteq \{(\sigma \downarrow L, X) \mid (\sigma, X) \in FP\}$ and $FP \uparrow L \doteq \{(\sigma', X) \mid \sigma' \in \sigma \uparrow L \wedge (\sigma, X) \in FP\}$.

**Corollary 6.17** Consider sets of failure pairs $FP$ and $FP'$ both defined on a set of labels $L$.

(a) If $FP = FP'$ then for an arbitrary $L'$, $FP \downarrow L' = FP' \downarrow L'$ and $FP \uparrow L' = FP' \uparrow L'$;

(b) For a set $L'$ disjoint from $L$, $(FP \uparrow L') \downarrow L' = FP$.

**Lemma 6.18** Consider sets of failure pairs $FP$ and $FP'$ both defined on a set of labels $L$ and a set $L'$ disjoint from $L$. $FP = FP'$ if and only if $FP \uparrow L' = FP' \uparrow L'$.

*Proof.* The implication from left to right follows trivially from the definition of $FP \uparrow L'$ (see the first item of Corollary 6.17). Then, it remains to prove $FP = FP'$ assuming the left-to-right implication and $FP \uparrow L' = FP' \uparrow L'$. It follows from the first item of Corollary 6.17 that $(FP \uparrow L') \downarrow L' = (FP' \uparrow L') \downarrow L'$ and from the second item of the same corollary, $FP = FP'$. ⊠

Consider a failure pair $(\sigma, X) \in FP(tss_0, p)$. This means that $\sigma \in TR(tss_0, p)$ and for some $p'$ such that $tss_0 \vDash p \xrightarrow{\sigma}{}^* p'$ and for all $l \in X$, $tss_0 \vDash p \overset{l}{\nrightarrow}$. Then, for all $\sigma' \in \sigma \uparrow L'$, $(\sigma', X)$ is a failure pair originating from $p$ in $tss_1$ since $\sigma' \in TR(tss_1, p)$ and the blocked transitions from $X$ remain blocked since $L \cap L' = \emptyset$ and $X \subseteq L$ and hence $X \cap L' = \emptyset$, i.e., the added $L'$-transitions do not change the status of $X$. Conversely, if a pair $(\sigma, X) \in FP(tss_1, p)$ then $(\sigma \downarrow L', X)$ is a failure pair of $p$ in $tss_0$ since first, $\sigma \downarrow L' \in TR(tss_0, p)$ (see the previous item) and second, $X$ may not contain any label from $L'$ since all transitions with a label from $L'$ are enabled in $tss_1$.

**Corollary 6.19** Let $tss_0$ and $tss_1$ be the TSS's defined above. For all $p \in \mathcal{C}(\Sigma_0)$, $FP(tss_1, p) = FP(tss_0, p) \uparrow L'$.

We now have $tss_0 \vDash p =_{\mathrm{F}} q$ iff, by definition, $FP(tss_0, p) = FP(tss_0, q)$ iff, by Lemma 6.18, $FP(tss_0, p) \uparrow L' = FP(tss_0, p) \uparrow L'$ iff, by Corollary 6.19, $FP(tss_1, p) = FP(tss_1, q)$ iff, by definition, $tss_1 \vDash p =_{\mathrm{F}} q$.

3. Ready equivalence: Similar to the previous two items; the only difference is that $L'$ is always a member of ready sets in $tss_1$ and should be removed from the ready sets when projecting from the ready traces in $tss_1$ to the ready traces in $tss_0$.

**Definition 6.20** Let $tss = (\Sigma, V, L, D)$ be a TSS. *A pair* $(\sigma, X) \in L^* \times \mathbb{P}(L)$ *is a ready pair of* $p \in \mathcal{C}$ *originating from* $tss$ if $tss \vDash p \xrightarrow{\sigma}{}^* p'$ for some $p'$ such that $X = \{l \in L \mid p' \xrightarrow{l'}\}$. The set of ready pairs in $tss$ originating from $p \in \mathcal{C}$, denoted by $R(tss, p)$ is the set containing all ready pairs of $p$ originating from $tss$. The processes $p$ and $q$ are ready equivalent w.r.t. TSS $tss$, notation $tss \vDash p =_{\mathrm{R}} q$, iff $R(tss, p) = R(tss, q)$.

For a set of ready pairs $R$ and a set of labels $L$, we define $R \downarrow L \doteq \{(\sigma \downarrow L, X \setminus L) \mid (\sigma, X) \in R\}$ and $R \uparrow L \doteq \{(\sigma', X \cup L) \mid \sigma' \in \sigma \uparrow L \wedge (\sigma, X) \in R\}$.

**Corollary 6.21** Consider sets of ready pairs $R$ and $R'$ both defined on a set of labels $L$.

(a) If $R = R'$ then for an arbitrary $L'$, $R \downarrow L' = R' \downarrow L'$ and $R \uparrow L' = R' \uparrow L'$;
(b) For a set $L'$ disjoint from $L$, $(R \uparrow L') \downarrow L' = R$.

**Lemma 6.22** Consider sets of ready pairs $R$ and $R'$ both defined on a set of labels $L$ and a set $L'$ disjoint from $L$. $R = R'$ if and only if $R \uparrow L' = R' \uparrow L'$.

*Proof.* The implication from left to right follows trivially from the definition of $R \uparrow L'$ (see the first item of Corollary 6.21). Then, it remains to prove $R = R'$ assuming the left-to-right implication and $R \uparrow L' = R' \uparrow L'$. It follows from the first item of Corollary 6.21 that $(R \uparrow L') \downarrow L' = (R' \uparrow L') \downarrow L'$ and from the second item of the same corollary, $R = R'$. ⊠

Consider a ready pair $(\sigma, X) \in R(tss_0, p)$. This means that $\sigma \in TR(tss_0, p)$ and for some $p'$ such that $tss_0 \vDash p \xrightarrow{\sigma}{}^* p'$ and $X = \{l \in L \mid p' \xrightarrow{l}\}$. Then, for all $\sigma' \in \sigma \uparrow L'$, $(\sigma', X \cup L') \in R(tss_1, p)$ since $\sigma' \in TR(tss_1, p)$ and the transitions from $X$ remain enabled and the transitions from $L'$ are also enabled since $tss_1$ is a granting extension of $tss_0$. Conversely, if a pair $(\sigma, X) \in R(tss_1, p)$ then $(\sigma \downarrow L', X \setminus L') \in R(tss_0, p)$.

**Corollary 6.23** Let $tss_0$ and $tss_1$ be the TSS's defined above. For all $p \in \mathcal{C}(\Sigma_0)$, $R(tss_1, p) = R(tss_0, p) \uparrow L'$.

We now have $tss_0 \vDash p =_{\mathrm{R}} q$ iff, by definition, $R(tss_0, p) = R(tss_0, q)$ iff, by Lemma 6.22, $R(tss_0, p) \uparrow L' = R(tss_0, p) \uparrow L'$ iff, by Corollary 6.23, $R(tss_1, p) = R(tss_1, q)$ iff, by definition, $tss_1 \vDash p =_{\mathrm{R}} q$.

4. Failure trace equivalence $=_{\mathrm{FT}}$: Similar to the above item; elements of $L'$ cannot be present in the refusal sets in the failure traces of $tss_0$ and hence one can repeat the above reasoning to prove the coincidence of failure traces.

**Definition 6.24** Let $tss = (\Sigma, V, L, D)$ be a TSS. The *refusal relation* $\dashrightarrow \subseteq \mathcal{C} \times \mathbb{P}(L) \times \mathcal{C}$ is defined as $p \overset{X}{\dashrightarrow} q$ iff $p = q$ and $p \overset{l}{\nrightarrow}$ for all $l \in X$. The *failure trace relation* $\rightarrowtail \subseteq \mathcal{C} \times (L \cup \mathbb{P}(L))^* \times \mathcal{C}$ is defined as the reflexive transitive closure of the transition relation $\rightarrow$ and the refusal relation $\dashrightarrow$. $\sigma \in (L \cup \mathbb{P}(L))^*$ is a *failure trace* of a process $p$ w.r.t. $tss$ if $p \overset{\sigma}{\rightarrowtail}$. Let $FT(tss, p)$ denote the failure traces of $p$ w.r.t. $tss$. The processes $p$ and $q$ are failure trace equivalent w.r.t. $tss$, notation $tss \vDash p =_{\mathrm{FT}} q$, iff $FT(tss, p) = FT(tss, q)$.

For a failure trace $\sigma$ on labels $L$, the failure trace $\sigma \downarrow L'$ is defined inductively by:

$$\begin{array}{lcl} \epsilon \downarrow L' & \doteq & \epsilon, \\ (l)\sigma \downarrow L' & \doteq & \begin{cases} \sigma \downarrow L' & l \in L', \\ (l)(\sigma \downarrow L') & l \notin L', \end{cases} \\ (X)\sigma \downarrow L' & \doteq & (X \setminus L')(\sigma \downarrow L'). \end{array}$$

For a set of failure traces $FT$, $FT \downarrow L' \doteq \{\sigma \downarrow L' \mid \sigma \in FT\}$. Given a failure trace $\sigma$ and a set of labels $L'$ *the granting extension of* $\sigma$ *with* $L'$, denoted by $\sigma \uparrow L'$, is the smallest set of traces that satisfies for all $\sigma_0, \sigma_1$ and for all $l \in L'$:

(a) $\sigma \in \sigma \uparrow L'$;

(b) $(\sigma_0)(\sigma_1) \in \sigma \uparrow L' \Rightarrow (\sigma_0)(l)(\sigma_1) \in \sigma \uparrow L'$.

For a set of failure traces $FT$, $FT \uparrow L' \doteq \bigcup_{\sigma \in FT} \sigma \uparrow L'$.

**Corollary 6.25** Consider sets of failure traces $FT$ and $FT'$ both defined on a set of labels $L$.

(a) If $FT = FT'$ then for an arbitrary $L'$, $FT \downarrow L' = FT' \downarrow L'$ and $FT \uparrow L' = FT' \uparrow L'$;

(b) For a set $L'$ disjoint from $L$, $(FT \uparrow L') \downarrow L' = FT$.

**Lemma 6.26** Consider sets of failure traces $FT$ and $FT'$ both defined on a set of labels $L$ and a set $L'$ disjoint from $L$. $FT = FT'$ if and only if $FT \uparrow L' = FT' \uparrow L'$.

*Proof.* The implication from left to right follows trivially from the definition of $FT \uparrow L'$ (see the first item of Corollary 6.25). Then, it remains to prove $FT = FT'$ assuming the left-to-right implication and $FT \uparrow L' = FT' \uparrow L'$. It follows from the first item of Corollary 6.25 that $(FT \uparrow L') \downarrow L' = (FT' \uparrow L') \downarrow L'$ and from the second item of the same corollary, $FT = FT'$. ⊠

**Corollary 6.27** Let $tss_0$ and $tss_1$ be the TSS's defined above. For all $p \in \mathcal{C}(\Sigma_0)$, $FT(tss_1, p) = FT(tss_0, p) \uparrow L'$.

We now have $tss_0 \vDash p =_{\mathrm{FT}} q$ iff, by definition, $FT(tss_0, p) = FT(tss_0, q)$ iff, by Lemma 6.26, $FT(tss_0, p) \uparrow L' = FT(tss_0, p) \uparrow L'$ iff, by Corollary 6.27, $FT(tss_1, p) = FT(tss_1, q)$ iff, by definition, $tss_1 \vDash p =_{\mathrm{FT}} q$.

5. Ready trace equivalence: Again similar to item 1 but the same observation as in item 3 should be noted along the ready traces.

   **Definition 6.28** Let $tss = (\Sigma, V, L, D)$ be a TSS. The *ready trace relation* $\rightsquigarrow \subseteq \mathcal{C} \times (L \cup \mathbb{P}(L))^* \times \mathcal{C}$ is defined recursively as follows: for $p, q, r \in \mathcal{C}$, $l \in L$ and $X \subseteq L$

   - $p \overset{\epsilon}{\rightsquigarrow} p$;
   - $p \overset{l}{\rightarrow} q$ implies $p \overset{(l)}{\rightsquigarrow} q$;
   - $p \overset{X}{\rightsquigarrow} p$ in case $X = \{l \in L \mid p \overset{l}{\rightarrow}\}$;
   - if $p \overset{\sigma}{\rightsquigarrow} q$ and $q \overset{\sigma'}{\rightsquigarrow} r$, then $p \overset{\sigma\sigma'}{\rightsquigarrow} r$.

   The sequence $\sigma \in (L \cup \mathbb{P}(L))^*$ is a *ready trace* of a process $p$ w.r.t. $tss$ if $p \overset{\sigma}{\rightsquigarrow}$. Let $RT(tss, p)$ denote the ready traces of $p$ w.r.t. $tss$. The processes $p$ and $q$ are ready trace equivalent w.r.t. $tss$, notation $tss \vDash p =_{\mathrm{RT}} q$, iff $RT(tss, p) = RT(tss, q)$.

   For a ready trace $\sigma$ on labels $L$, the ready trace $\sigma \downarrow L'$ is defined inductively by:

$$
\begin{array}{lcl}
\epsilon \downarrow L' & \doteq & \epsilon, \\
(l)\sigma \downarrow L' & \doteq & \begin{cases} \sigma \downarrow L' & l \in L', \\ (l)(\sigma \downarrow L') & l \notin L', \end{cases} \\
(X)\sigma \downarrow L' & \doteq & (X \setminus L')(\sigma \downarrow L').
\end{array}
$$

   For a set of ready traces $RT$, $RT \downarrow L' \doteq \{\sigma \downarrow L' \mid \sigma \in RT\}$. Given a ready trace $\sigma$ and a set of labels $L'$ *the granting extension of* $\sigma$ *with* $L'$, denoted by $\sigma \uparrow L'$, is the smallest set of traces that satisfies $\forall_{\sigma_0, \sigma_1}$ and $\forall_{l \in L'}$:

   (a) $\sigma \in (\sigma \sqcup L') \uparrow L'$;
   (b) $(\sigma_0)(\sigma_1) \in \sigma \uparrow L' \Rightarrow (\sigma_0)(l)(\sigma_1) \in \sigma \uparrow L'$;

where $\sqcup$ is defined inductively by:

$$\epsilon \sqcup L \doteq \epsilon, \qquad ((l)\sigma) \sqcup L \doteq (l)(\sigma \sqcup L), \qquad ((X)\sigma) \sqcup L \doteq (X \cup L)(\sigma \sqcup L).$$

For a set of ready traces $RT$, $RT \uparrow L' \doteq \bigcup_{\sigma \in RT} \sigma \uparrow L'$.

**Corollary 6.29** Consider sets of ready traces $RT$ and $RT'$ both defined on a set of labels $L$.

(a) If $RT = RT'$ then for an arbitrary $L'$, $RT \downarrow L' = RT' \downarrow L'$ and $RT \uparrow L' = RT' \uparrow L'$;

(b) For a set $L'$ disjoint from $L$, $(RT \uparrow L') \downarrow L' = RT$.

**Lemma 6.30** Consider sets of ready traces $RT$ and $RT'$ both defined on a set of labels $L$ and a set $L'$ disjoint from $L$. $RT = RT'$ if and only if $RT \uparrow L' = RT' \uparrow L'$.

*Proof.* The implication from left to right follows trivially from the definition of $RT \uparrow L'$ (see the first item of Corollary 6.29). Then, it remains to prove $RT = RT'$ assuming the left-to-right implication and $RT \uparrow L' = RT' \uparrow L'$. It follows from the first item of Corollary 6.29 that $(RT \uparrow L') \downarrow L' = (RT' \uparrow L') \downarrow L'$ and from the second item of the same corollary, $RT = RT'$. ⊠

**Corollary 6.31** Let $tss_0$ and $tss_1$ be the TSS's defined above. For all $p \in \mathcal{C}(\Sigma_0)$, $RT(tss_1, p) = RT(tss_0, p) \uparrow L'$.

We now have $tss_0 \vDash p =_{\mathrm{RT}} q$ iff, by definition, $RT(tss_0, p) = RT(tss_0, q)$ iff, by Lemma 6.30, $RT(tss_0, p) \uparrow L' = RT(tss_0, p) \uparrow L'$ iff, by Corollary 6.31, $RT(tss_1, p) = RT(tss_1, q)$ iff, by definition, $tss_1 \vDash p =_{\mathrm{RT}} q$.

6. Simulation equivalence $\rightleftarrows$:

**Definition 6.32** Let $tss = (\Sigma, V, L, D)$ be a TSS. A *simulation* w.r.t. $tss$ is a binary relation $R$ on processes, satisfying, for $l \in L$: if $pRq$ and $tss \vDash p \xrightarrow{l} p'$, then $q \xrightarrow{l} q'$ and $p'Rq'$ for some $q'$. The processes $p$ and $q$ are simulation equivalent or similar w.r.t. TSS $tss$, notation $tss \vDash p \rightleftarrows q$, iff there exists a simulation $R$ such that $pRq$ and a simulation $R'$ such that $qR'p$.

Suppose that $tss_0 \vDash p \rightleftarrows q$. Then there exist simulations $R$ and $R'$ such that $pRq$ and $qR'p$. It is very easy to check that $R \cup Id$ and $R' \cup Id$ with $Id = \{(p,p) \mid p \in \mathcal{C}(\Sigma_0)\}$ are simulations with respect to $tss_1$ and hence, $tss_1 \vDash p \rightleftarrows q$.

For the inclusion in the other direction, suppose that $tss_1 \models p \rightleftarrows q$. Then, there exist simulations $R$ and $R'$ w.r.t. $tss_1$ such that $pRq$ and $qR'p$. One can easily verify that $R \cap (\mathcal{C}(\Sigma_0) \times \mathcal{T}(\Sigma_0))$ and $R' \cap (\mathcal{C}(\Sigma_0) \times \mathcal{C}(\Sigma_0))$ are both simulations w.r.t. $tss_0$ and hence $tss_0 \models p \rightleftarrows q$.

7. Ready simulation equivalence $=_{\mathrm{RS}}$:

   **Definition 6.33** Let $tss = (\Sigma, V, L, D)$ be a TSS. A *ready simulation* w.r.t. $tss$ is a binary relation $R$ on processes, satisfying, for $l \in L$: (1) if $pRq$ and $tss \models p \xrightarrow{l} p'$, then $q \xrightarrow{l} q'$ and $p'Rq'$ for some $q'$, and (2) if $pRq$, then $p \xrightarrow{l'}$ iff $q \xrightarrow{l'}$. The processes $p$ and $q$ are ready simulation equivalent or ready similar w.r.t. TSS $tss$, notation $tss \models p =_{\mathrm{RS}} q$, iff there exist a ready simulation $R$ such that $pRq$ and a ready simulation $R'$ such that $qR'p$.

   Suppose that $tss_0 \models p =_{\mathrm{RS}} q$. Then there exist ready simulations $R$ and $R'$ such that $pRq$ and $qR'p$. It is very easy to check that $R \cup Id$ and $R' \cup Id$ with $Id = \{(p,p) \mid p \in \mathcal{C}(\Sigma_0)\}$ are ready simulations with respect to $tss_1$ and hence, $tss_1 \models p =_{\mathrm{RS}} q$.

   For the inclusion in the other direction, suppose that, $tss_1 \models p =_{\mathrm{RS}} q$. Then, there exist ready simulations $R$ and $R'$ w.r.t. $tss_1$ such that $pRq$ and $qR'p$. One can easily verify that $R \cap (\mathcal{C}(\Sigma_0) \times \mathcal{C}(\Sigma_0))$ and $R' \cap (\mathcal{C}(\Sigma_0) \times \mathcal{C}(\Sigma_0))$ are both ready simulations w.r.t. $tss_0$ and hence $tss_0 \models p =_{\mathrm{RS}} q$.

8. Weak bisimulation equivalence $\underline{\leftrightarrow}_{\mathrm{w}}$:

   **Definition 6.34** Let $tss = (\Sigma, V, L, D)$ be a TSS. A *weak bisimulation* w.r.t. $tss$ is a symmetric binary relation $R$ on processes, satisfying, for $l \in L$: if $pRq$ and $tss \models p \xrightarrow{l} p'$, then either $l = \tau$ and $p'Rq$, or $q \Longrightarrow q_1 \xrightarrow{l} q_2 \Longrightarrow q'$ and $p'Rq'$ for some $q_1, q_2, q'$. Here $\Longrightarrow$ denotes the reflexive transitive closure of $\xrightarrow{\tau}$. The processes $p$ and $q$ are weakly bisimulation equivalent or weakly bisimilar w.r.t. TSS $tss$, notation $tss \models p \underline{\leftrightarrow}_{\mathrm{w}} q$, iff there exist a weak bisimulation $R$ such that $pRq$.

   Suppose that for $p, q \in \mathcal{C}(\Sigma_0)$, $tss_0 \models p \underline{\leftrightarrow}_{\mathrm{w}} q$. Then there exists a weak bisimulation $R$ such that $pRq$. We claim that $R \cup Id$ (for $Id$ defined in the previous item) is a weak bisimulation w.r.t. $tss_1$. It is trivial to see that the pairs from $Id$ respect the requirements for a weak bisimulation. So it remains to verify this for pairs $pRq$. In $tss_1$, all transitions with a label from $L_0$ are accounted for by $R$ since $tss_1$ is a granting extension of $tss_0$ which means that $tss_0 \models p \xrightarrow{l} p'$ iff $tss_1 \models p \xrightarrow{l} p'$. For the new transitions, $p \xrightarrow{l} p'$ for some $l \in L'$, we have that $p = p'$. Again, since the extension is granting we also have $q \xrightarrow{l} q$. Thus we have $q \Longrightarrow q \xrightarrow{l} q \Longrightarrow q$ and $pRq$.

Suppose that $tss_1 \models p \underline{\leftrightarrow}_{\mathrm{w}} q$. Then there exists a weak bisimulation $R$ such that $pRq$. We claim that $R' = R \cap (\mathcal{C}(\Sigma_0) \times \mathcal{C}(\Sigma_0))$ is a weak bisimulation w.r.t. $tss_0$. The crucial observation is that since the extension is granting, no transitions from a $\mathcal{C}(\Sigma_0)$ term to $\mathcal{C}(\Sigma_1)$ term are possible. Then, obviously every pair from $R'$ satisfies the requirements of a weak bisimulation.

9. Rooted weak bisimulation equivalence $\underline{\leftrightarrow}_{\mathrm{rw}}$:

   **Definition 6.35** Let $tss = (\Sigma, V, L, D)$ be a TSS. The processes $p$ and $q$ are rooted weak bisimulation equivalent or rooted weak bisimilar w.r.t. TSS $tss$, notation $tss \models p \underline{\leftrightarrow}_{\mathrm{rw}} q$, iff there exist a weak bisimulation $R$ such that $pRq$ and whenever $p \xrightarrow{l} p'$ there exists $q_1, q_2, q'$ such that $q \Longrightarrow q_1 \xrightarrow{l} q_2 \Longrightarrow q'$ and $p'Rq'$ and whenever $q \xrightarrow{l} q'$ there exists $p_1, p_2, p'$ such that $p \Longrightarrow p_1 \xrightarrow{l} p_2 \Longrightarrow p'$ and $p'Rq'$.

   Apart from the proof given in item 8, it remains to show that for two terms $p, q \in \mathcal{C}(\Sigma_0)$, the root condition holds w.r.t. $tss_0$ if and only if it holds for $tss_1$. For all labels $l$ from $L_0$ this follows immediately from the fact that $tss_1$ is a granting extension of $tss_0$. For labels $l'$ from $L'$, the only relevant (new) transitions in $tss_1$ are $p \xrightarrow{l'} p$ and $q \xrightarrow{l'} q$. Obviously, they pose no problem.

10. Branching bisimulation equivalence $\underline{\leftrightarrow}_{\mathrm{b}}$:

    **Definition 6.36** Let $tss = (\Sigma, V, L, D)$ be a TSS. A *branching bisimulation* w.r.t. $tss$ is a symmetric binary relation $R$ on processes, satisfying, for $l \in L$: if $pRq$ and $tss \models p \xrightarrow{l} p'$, then either $l = \tau$ and $p'Rq$, or $q \Longrightarrow q_1 \xrightarrow{l} q'$ and $pRq_1$ and $p'Rq'$ for some $q_1, q_2, q'$. The processes $p$ and $q$ are branching bisimulation equivalent or branching bisimilar w.r.t. TSS $tss$, notation $tss \models p \underline{\leftrightarrow}_{\mathrm{b}} q$, iff there exist a branching bisimulation $R$ such that $pRq$.

    The proof is similar to the proof of item 8.

11. Rooted branching bisimulation equivalence $\underline{\leftrightarrow}_{\mathrm{rb}}$:

    **Definition 6.37** Let $tss = (\Sigma, V, L, D)$ be a TSS. The processes $p$ and $q$ are rooted branching bisimulation equivalent or rooted branching bisimilar w.r.t. TSS $tss$, notation $tss \models p \underline{\leftrightarrow}_{\mathrm{rb}} q$, iff there exist a branching bisimulation $R$ such that $pRq$ and whenever $p \xrightarrow{l} p'$ there exists $q'$ such that $q \xrightarrow{l} q'$ and $p'Rq'$ and whenever $q \xrightarrow{l} q'$ there exists $p'$ such that $p \xrightarrow{l} p'$ and $p'Rq'$.

    The proof is similar to the proof of the previous item.

12. Bisimulation equivalence $\underline{\leftrightarrow}$ :

**Definition 6.38** Let $tss = (\Sigma, V, L, D)$ be a TSS. A *bisimulation* w.r.t. $tss$ is a binary relation $R$ on processes, satisfying, for $l \in L$: (1) if $pRq$ and $tss \vDash p \xrightarrow{l} p'$, then $q \xrightarrow{l} q'$ and $p'Rq'$ for some $q'$, if $pRq$ and $tss \vDash q \xrightarrow{l} q'$, then $p \xrightarrow{l} p'$ and $p'Rq'$ for some $p'$. The processes $p$ and $q$ are bisimulation equivalent w.r.t. TSS $tss$, notation $tss \vDash p \underline{\leftrightarrow} q$, iff there exist a bisimulation $R$ such that $pRq$.

Suppose that $tss_0 \vDash p \underline{\leftrightarrow} q$. Then there exist a bisimulation $R$ such that $pRq$. It is very easy to check that $R \cup Id$ is a bisimulation with respect to $tss_1$ and hence, $tss_1 \vDash p \underline{\leftrightarrow} q$.

For the inclusion in the other direction, suppose that, $tss_1 \vDash p \underline{\leftrightarrow} q$. Then, there exists a bisimulation $R$ w.r.t. $tss_1$ such that $pRq$. One can easily verify that $R \cap (\mathcal{C}(\Sigma_0) \times \mathcal{C}(\Sigma_0))$ is a bisimulation w.r.t. $tss_0$ and hence $tss_0 \vDash p \underline{\leftrightarrow} q$.

⊠

Using the result of Theorem 6.10, henceforth, we refer to the concept of *orthogonality* and by that we mean $\sim$-orthogonality with respect to any of the notions of behavioral equivalence named above.

Unfortunately, not all notions of behavioral equivalence are preserved under granting extensions, i.e., granting extensions are not $\sim$-orthogonal for *all* notions of behavioral equivalence $\sim$. The only two counterexamples that we encountered so far in the literature are the notions of *completed-trace equivalence* and *complete simulation equivalence*. Next, we give a counterexample for completed-trace equivalence. The same example is also a counterexample for complete simulation equivalence.

**Example 6.39** Consider the following deduction rule added to the semantics of *MPA* ($tss_m$ of Example 6.2) which we refer to as $MPA_\omega$.

$$\frac{}{a^\omega \xrightarrow{a} a^\omega}$$

Let $=_{\mathrm{CT}}$ denote completed trace equivalence. It does not hold that $a^\omega + a.0 =_{\mathrm{CT}} a^\omega$ since $a^\omega + a.0$ has a completed trace $a$ while $a^\omega$ has no completed trace. (Note that the set of traces of these two process are equal, i.e., $a^\omega + a.0$ and $a^\omega$ are trace equivalent.)

Consider the granting extension of $MPA_\omega$ with the following deduction rule.

$$\frac{}{x \xrightarrow{1} x}$$

For $MPA_\omega$ extended with the above rule, the two processes become completed-trace equivalent, since both do not have any completed trace anymore.

## 6.4 Orthogonality Meta-Theorems

### 6.4.1 Preliminaries

In this section, we seek sufficient conditions for establishing orthogonality and equational conservativity.

As stated in Chapter 2, different interpretations of the transition relation, i.e., the set of closed (positive) formulae, induced by a TSS are given in the literature. In this section, we formulate and prove our main results in such a general way that they remain independent from the chosen interpretation and can be adopted for several existing ones. In cases where we need an explicit transition relation, we assume that this transition relation is uniquely defined by the corresponding TSS using one of the interpretations given in [59]. In such cases, we use the notation $tss \vDash \phi$ to denote that a closed positive formula $\phi$ is in the transition relation induced by $tss$. If we need to go further and examine the proof of a formula in a TSS, we use the following notion of *provable transition rules*.

**Definition 6.40 (Provable Transition Rules)** A deduction rule $\frac{N}{c}$ is called a *transition rule* if $c$ is a closed positive formula and $N$ is a set of closed negative formulae.

A transition rule $\frac{N}{c}$ is provable with respect to $tss$, denoted by $tss \vdash \frac{N}{c}$ if and only if there exists a well-founded upwardly branching proof tree with nodes labelled by formulae such that:

- the root of the proof tree is labelled by $c$;
- if the label of a node $q$, denoted by $\psi$, is a positive formula and $\{\psi_i \mid i \in I\}$ is the set of labels of the nodes directly above $q$, then there is a deduction rule $\frac{\{\chi_i \mid i \in I\}}{\chi}$ in $tss$ (N.B. $\chi_i$ can be a positive or a negative formula) and a substitution $\sigma$ such that $\sigma(\chi) = \psi$ and for all $i \in I$, $\sigma(\chi_i) = \psi_i$;
- $\chi \in N$ for all negative formulae $\chi$, if and only if $\chi$ is a leaf of the proof tree.

This technique will enable us to apply our results to various existing interpretations (following [48]). Particularly, if $tss \vdash \frac{}{\phi}$ and $tss$ induces a unique transition relation using one of the existing interpretations, it always follows that $tss \vDash \phi$.

### 6.4.2 Granting Meta-Theorems

We start with defining sufficient conditions to prove an extension to be granting. Hence, we need to define when a deduction rule proves (only) self-transitions. We use unification as a means to this end.

**Definition 6.41 (Unification)** A term $t$ is *unifiable* with $t'$ using $\sigma$, denoted by $t \approx_\sigma t'$ if and only if $\sigma(t) = \sigma(t')$. The set of unifiers of $t$ and $t'$ is defined by $U(t,t') = \{\sigma \mid t \approx_\sigma t'\}$. The set of unifiers of a set of pairs is defined as the intersection of the sets of unifiers of each pair. The set of unifiers of an empty set is defined to include all substitutions. The set of unifiers of a positive formula $t \xrightarrow{l} t'$ is defined as the set of unifiers of $t$ and $t'$. Unification also naturally extends to a set of positive formulae, again, using intersection.

Next, we characterize the set of rules that induce self-transitions. This is done by only allowing for unifiable (positive) formulae in the premises and the conclusion of a rule and further, by forcing the unification of the conclusion to follow from that of the premises.

**Definition 6.42** (Source-Preserving Rules) A deduction rule $\frac{H}{c}$ without negative premises is *source preserving* if $U(H) \neq \emptyset$ and $U(H) \subseteq U(c)$. A TSS is *source preserving* if all its deduction rules are. For a source-preserving TSS, the set of *unified-conclusions* contains conclusions of the deduction rules with their unifiers applied to them.

The following lemma captures the intuition behind source-preserving rules.

**Lemma 6.43** If $tss$ is source preserving, then $\forall_{l \in L} \forall_{p,p' \in \mathcal{C}}\ tss \vDash p \xrightarrow{l} p' \Rightarrow p = p'$.

*Proof.* Source-preserving rules do not contain negative premises and hence the two notations $tss \vDash p \xrightarrow{l} p'$ and $tss \vdash \frac{}{p \xrightarrow{l} p'}$ coincide. Hence, we proceed with an induction on the depth of the proof for $tss \vdash \frac{}{p \xrightarrow{l} p'}$.

If the proof tree has depth one, then the transition is due to a rule of the following form

$$\frac{}{t \xrightarrow{l} t'}$$

and a substitution $\sigma$ such that $\sigma(t) = p$ and $\sigma(t') = p'$. But since $tss$ is source preserving, it holds that $U(\emptyset) \subseteq U(t \xrightarrow{l} t')$ and since $\sigma \in U(\emptyset)$, it follows that $\sigma \in U(t \xrightarrow{l} t')$ and hence, $p = \sigma(t) = \sigma(t') = p'$.

For the induction step, suppose that the last deduction rule in the proof tree of depth $n$ has the following form

$$\frac{H}{t \xrightarrow{l} t'}$$

and there exists a substitution $\sigma$ such that $\sigma(t) = p$ and $\sigma(t') = p'$ and for all $h \in H$, $\sigma(h)$ is provable, trivially with a proof of depth less than $n$. Following the induction hypothesis, for all $h \in H$, $\sigma \in U(h)$ and thus, $\sigma \in U(H)$. It then follows from $U(H) \subseteq U(t \xrightarrow{l} t')$ that $\sigma \in U(t \xrightarrow{l} t')$ and we conclude that $p = \sigma(t) = \sigma(t') = p'$. $\boxtimes$

As illustrated above, source-preserving rules are safe for the purpose of proving self-transitions. However, there might be other rules in the extending TSS that can be harmful in that they may prove other types of transition for old terms. This may be prevented by forcing the other (non source-preserving) rules to have negative or non-unifiable positive premises addressing the old syntax. The following definitions give sufficient conditions for an extension to be granting.

**Definition 6.44 (Generated Terms)** The set of terms *generated* by a set of terms $S$, denoted by $G(S)$, is the set of all terms $t' = \sigma(t)$, for some $t \in S$ and some $\sigma$ such that $\forall_{x \in V} \sigma(x) \in S$. A set of terms $S$ *covers* $\Sigma$-terms, if $\mathcal{C} \subseteq G(S)$.

**Definition 6.45 (Granting Criteria)** Consider a TSS $tss = (\Sigma, V, L, D)$ stratified by $\mathcal{S}$. It *grants* $L_0$ *transitions on* $\Sigma_0$*-terms*, if $tss = tss_0 \cup tss_1$ (with $tss_x = (\Sigma_x, V, L_x, D_x)$ for $x \in \{0, 1\}$) such that:

1. $tss_0$ is strictly stratified by $\mathcal{S}$, it is source dependent and for all $l \in L_0$, the set containing sources of unified-conclusions of $l$-rules covers $\Sigma_0$-terms, and
2. for all deduction rules $d \in D_1$ at least one of the following holds:
   (a) $d$ has a function symbol from $\Sigma_1 \setminus \Sigma_0$ in the source of its conclusion, or
   (b) $\rho(d, \Sigma_0)$ has a source-dependent negative premise with a label in $L_1$, or
   (c) $\rho(d, \Sigma_0)$ has a source-dependent positive premise $t \xrightarrow{l} t'$ with $l \in L_1$ and $U(t, t') = \emptyset$.

The first condition in the above definition is dedicated to proving self-transitions from the syntax of $\Sigma_0$, and the second one takes care of preventing $\Sigma_0$-terms from performing other types of transitions while allowing other terms to do so.

**Theorem 6.46** (Granting Meta-theorem) Consider TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$. If $tss_0$ is source dependent, $tss_1$ grants $L_1$ transitions on $\Sigma_0$-terms and $L_0 \cap L_1 = \emptyset$ then $tss_0 \cup tss_1$ is a granting extension of $tss_0$.

*Proof.* Since $tss_1$ grants $L_1$ transitions over $\Sigma_0$-terms, there exists a decomposition of $tss_1$ into $tss_A \cup tss_B$ such that $tss_A = (\Sigma_0, V, L_1, D_A)$ and $tss_B = (\Sigma_B, V, L_B, D_B)$ and

1. $tss_A$ is strictly stratified by $\mathcal{S}$, it is source preserving and for all $l \in L_1$, the set containing sources of unified-conclusions of $l$-rules covers $\Sigma_0$-terms, and
2. for all deduction rules in $d \in D_B$ at least one of the following holds:
   (a) $d$ has a function symbol from $\Sigma_B \setminus \Sigma_0$ in the source of its conclusion, or
   (b) $\rho(d, \Sigma_0)$ has a source-dependent negative premise with a label in $L_1$, or
   (c) $\rho(d, \Sigma_0)$ has a source-dependent positive premise $t \xrightarrow{l} t'$ with $l \in L_1$ and $U(t, t') = \emptyset$.

To show that $tss_0 \cup tss_1$ is a granting extension of $tss_0$, we have to prove the following two items:

1. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)} \forall_{l \in L_0} tss_0 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_0 \cup tss_1 \vDash p \xrightarrow{l} p'$. To show this, we prove that the sets of provable transition rules with conclusion $p \xrightarrow{l} p'$ w.r.t. $tss_0$ and w.r.t. $tss_0 \cup tss_1$ coincide.[1]

   First, if a transition rule $\dfrac{N}{p \xrightarrow{l} p'}$ is provable w.r.t. $tss_0$, then the same proof tree can be used w.r.t. $tss_0 \cup tss_1$ to provide us with $tss_0 \cup tss_1 \vdash \dfrac{N}{p \xrightarrow{l} p'}$.

   Second, to prove that $tss_0 \cup tss_1 \vdash \dfrac{N}{p \xrightarrow{l} p'} \Rightarrow tss_0 \vdash \dfrac{N}{p \xrightarrow{l} p'}$, we prove the following stronger claim.

   **Claim.** If for some $p \in \mathcal{C}(\Sigma_0)$, $p' \in \mathcal{C}(\Sigma_0 \cup \Sigma_1)$, $l \in L_0$, $tss_0 \cup tss_1 \vdash \dfrac{N}{p \xrightarrow{l} p'}$ then $p' \in \mathcal{C}(\Sigma_0)$ and $tss_0 \vdash \dfrac{N}{p \xrightarrow{l} p'}$ (thus, $N$ is built upon $\Sigma_0$-terms and $L_0$ labels).

   *Proof.* Take an arbitrary term $p \in \mathcal{C}(\Sigma_0)$, we prove by an induction on the depth of the proof tree for $tss_0 \cup tss_1 \vdash \dfrac{N}{p \xrightarrow{l} p'}$ that $p' \in \mathcal{C}(\Sigma_0)$ and $tss_0 \vdash \dfrac{N}{p \xrightarrow{l} p'}$. If the proof tree has depth one, then it is due to a deduction rule $d = (H, c)$ in $tss_0$ with no positive premises (rules in $tss_1$ have a disjoint set of labels and thus cannot provide a proof for a transition with label $l \in L_0$) and a substitution $\sigma$. Since $d$ is source dependent and has no positive premise, variables in $H$ are all among the variables in the source of $c$. Thus, applying $\sigma$ to $H$ and $c$ yields terms from $\Sigma_0$. Hence, using deduction rule $d$ and

[1] This is similar to the technique used in [48] for proving equality of the induced transition relations of two TSS's. Here, however, this technique is used to show the equality of partitions of the induced transition relation.

substitution $\sigma$, we have a proof for $tss_0 \vdash \dfrac{N}{p \xrightarrow{l} p'}$ and this concludes the induction basis.

For the induction step, suppose that $tss_0 \cup tss_1 \vdash \dfrac{N}{p \xrightarrow{l} p'}$ has a proof of depth $n$. Then, the last deduction rule applied in the proof tree is in $tss_0$ (again, due to the disjointness of labels). Hence, the induction hypothesis applies to the positive premises (if any). Since $tss_0$ is source dependent, we can define a measure of source distance on premises as follows.

The source-dependency graph of a deduction rule is constructed by taking the variables that appear in the deduction rule as nodes and by putting an edge between two variables if one appears in the source and the other in the target of a premise. The distance of a variable in the source-dependency graph is the length of the shortest backward chain in this graph starting from the variable and ending in a variable in the source of the conclusion. The distance of a premise is the maximal distance of the variables of its source.

We proceed by an induction on the distance of premises in the source-dependency graph. For the induction basis, all the variables in the source of the premise should be among the variables in the source of the conclusion, hence the induction hypothesis on the depth of the proof applies and thus, the target of the premise (after substitution) should also be in $\mathcal{C}(\Sigma_0)$. Similarly, for the induction step, all the variables in the source of the premise under consideration should already be valuated by terms in $\mathcal{C}(\Sigma_0)$ and again by applying the induction hypothesis on the depth of the proof, we get that the variables in target are also valuated by terms in $\mathcal{C}(\Sigma_0)$. Hence, the premises are all provable from the same set of negative premises in $tss_0$ and this concludes the proof of the lemma. ⊠

2. $\forall_{p \in \mathcal{C}(\Sigma_0)} \forall_{p' \in \mathcal{C}(\Sigma_0 \cup \Sigma_1)} \forall_{l \in L_1}\ tss_0 \cup tss_1 \vdash p \xrightarrow{l} p' \Leftrightarrow p = p'$. We first prove the implication from right to left, which will be used in the proof of the implication in the other direction.

   Consider a formula $p \xrightarrow{l} p'$. $tss_A$ is source preserving and for all $l \in L_1$, the set of sources of unified-conclusions of $l$-rules cover $\Sigma_0$-terms. Hence, there should exist an $l$-rule in $D_A$ of the following form:

$$\frac{H}{t \xrightarrow{l} t'},$$

   and a unifier $\sigma$ of $t$ and $t'$ and a substitution $\sigma'$ such that $\sigma'(\sigma(t)) = p = \sigma'(\sigma(t'))$.

We proceed by an induction on the strict stratification $\mathcal{S}(p \xrightarrow{l} p)$. For the induction basis, consider $p \xrightarrow{l} p$ that has a minimal stratification measure. This means that deduction rule $d$ has no premises, since otherwise, the premises would also be unifiable using $\sigma \circ \sigma'$ and have a strictly smaller measure. For a rule with an empty set of premises, any substitution is a unifier for the conclusion and hence using $d$ and substitution $\sigma \circ \sigma'$, we have a proof for $p \xrightarrow{l} p$. For the induction step, since the unifier of the conclusion is a unifier for all the premises, by applying $\sigma \circ \sigma'$ on the premises and applying the induction hypothesis we construct a proof for $p \xrightarrow{l} p$ (see the proof of Lemma 6.43 for details).

For the implication in the other direction, namely $tss_0 \cup tss_1 \vdash p \xrightarrow{l} p' \Rightarrow p = p'$, we prove the following claim.

**Claim.** For all set of negative premises $N$ such that $\forall_{l \in L_1}\ p \xrightarrow{l}\!\!\!\!\!/\ \notin N$, $\forall_{l' \in L_1}$

$$tss_0 \cup tss_1 \vdash \frac{N}{p \xrightarrow{l'} p'} \Rightarrow p = p' \wedge N = \emptyset.$$

In the above claim, we require that $\forall_{l \in L_1}\ p \xrightarrow{l}\!\!\!\!\!/\ \notin N$ since in the proof of the implication from right to left, we have shown that these negative formulae can always be contradicted by constructing a proof for $p \xrightarrow{l} p$.

*Proof.* To prove the claim, we use an induction on the proof depth for $tss_0 \cup tss_1 \vdash \dfrac{N}{p \xrightarrow{l} p'}$.

For the induction basis, since $l' \in L_1$, there exist a deduction rule $d \in D_1$ with no positive premises, of the following form

$$\frac{H}{t \xrightarrow{l'} t'}$$

and a substitution $\sigma$ such that $\sigma(t) = p$ and $\sigma(t') = p'$. Suppose that $d$ is in $D_B$. Then it either has a function symbol not in $\Sigma_B \setminus \Sigma_0$ in the source of its conclusion or $\rho(d, \Sigma_0)$ has a negative premise $t_j \xrightarrow{l_j}\!\!\!\!\!/$ with $l_j \in L_1$. In the former case, the source of rule $d$ cannot match $p$. In the latter case, $vars(t_j) \subseteq vars(t)$ (since $d$ is source dependent and has no positive premises) and hence $\sigma(t_j) \in \mathcal{C}(\Sigma_0)$. Thus, there is a negative formula $\sigma(t_j) \xrightarrow{l_j}\!\!\!\!\!/\ \in N$, with $\sigma(t_j) \in \mathcal{C}(\Sigma_0)$ and $l_j \in L_1$, contradicting our hypothesis. From these two contradictions, we conclude that $d \in D_A$. Following the same reasoning as in the other implication, from a rule in $D_A$ with empty premises (since $d$ is source preserving and has no positive premises), we can only prove transition rules of the form $\dfrac{}{p \xrightarrow{l'} p}$.

For the induction step, there should again be a deduction rule $d$ in $D_1$ of the following form

$$\frac{H}{t \xrightarrow{l'} t'}$$

and a substitution $\sigma$ such that $\sigma(t) = p$. If $d \in D_B$ one of the following three should hold:

(a) $t \notin \mathcal{T}(\Sigma_0, V)$, then $\sigma(t) \notin \mathcal{C}(\Sigma_0)$ and this contradicts $\sigma(t) = p$;

(b) $\rho(d, \Sigma_0)$ has a negative premise $t_j \overset{l_j}{\nrightarrow}$ with $l_j \in L_1$. Then, using an induction on the chain of source dependencies leading to the premise $t_j \overset{l_j}{\nrightarrow}$, it follows that $\sigma(t_j) \in \mathcal{C}(\Sigma_0)$ and since $\sigma(t_j) \overset{l_j}{\nrightarrow} \in N$, it contradicts our hypothesis.

(c) $\rho(d, \Sigma_0)$ has a positive premise $t_i \xrightarrow{l_i} t_i'$ and $U(t_i \xrightarrow{l_i} t_i') = \emptyset$. Again by an induction on the chain of source dependencies, it follows that $\sigma(t_i) \in \mathcal{C}(\Sigma_0)$ and hence, following the induction hypothesis (concerning the proof depth), $\sigma(t_i) = \sigma(t_i')$, and this contradicts $U(t_i \xrightarrow{l_i} t_i') = \emptyset$.

Hence, we conclude that $d \in D_A$ and hence, it only has positive premises. We apply the induction hypothesis on all the premises in $H$ and conclude that $\sigma$ is a unifier for all $h \in H$ and hence it is a unifier for $t$ and $t'$. Furthermore, it follows from the induction hypothesis that the set of leaves for the proof tree above each and every premise is empty and hence the whole proof tree has no negative premise as a leaf. Hence for this proof, it holds that $N = \emptyset$ and $\sigma(t) = \sigma(t') = p$. ⊠

This completes the proof. ⊠

### 6.4.3 Decomposing Orthogonality

An operationally conservative extension denies all types of new behavior from the old syntax. In total contrast, a granting extension forces to add all types of behavior to the old syntax. It would be most interesting, if we could reach a compromise between these two types of extensions while preserving the orthogonality. Hence, in this section, we propose a few ways of decomposing orthogonality into operationally conservative and granting extensions.

The first way to decompose orthogonality is by taking two subsets of the extending TSS with disjoint sets of labels and proving that one subset is a conservative extension and the other set is a granting extension of the extended TSS.

**Theorem 6.47** Consider a TSS $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$. If $tss_1 = tss_{10} \cup tss_{11}$ (with $tss_x = (\Sigma_x, V, L_x, D_x)$ for $x \in \{10, 11\}$) such that:

1. $tss_{10}$ satisfies the operational conservativity criteria of Theorem 3.15 w.r.t. $tss_0$,
2. $tss_{11}$ satisfies the granting extension criteria of Theorem 6.46 w.r.t. $tss_0$, and
3. $L_{10} \cap L_{11} = \emptyset$,

then $tss_0 \cup tss_1$ is an orthogonal extension of $tss_0$.

*Proof.* Note that we cannot use Theorems 3.15 and 6.46 here directly by proving, for example, that $tss_0 \cup tss_1$ is a conservative extension of $tss_0 \cup tss_{11}$ since the hypotheses of Theorem 3.15 may be violated due to the addition of the new signature $\Sigma_{11}$.

Instead, we prove that $tss_0 \cup tss_1$ is a granting extension of $tss_A = (\Sigma_0, V, L_0 \cup L_{10}, D_0)$. Then $tss_0 \cup tss_1$ will be an orthogonal extension of $tss_0$ as well since the transition relations induced by $tss_0$ and $tss_A$ coincide.

To prove that $tss_0 \cup tss_1$ is a granting extension of $tss_A$, copying Definition 6.9, we have to prove:

1. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)} \forall_{l \in L_0 \cup L_{10}} tss_A \vdash p \xrightarrow{l} p' \Leftrightarrow tss_0 \cup tss_1 \vdash p \xrightarrow{l} p'$, and
2. $\forall_{p \in \mathcal{C}(\Sigma_0)} \forall_{p' \in \mathcal{C}(\Sigma_0 \cup \Sigma_1)} \forall_{l \in L_{11}} tss_0 \cup tss_1 \vdash p \xrightarrow{l} p' \Leftrightarrow p = p'$.

To prove the first item, we prove the following stronger claim.

**Claim.** $\forall_{p \in \mathcal{C}(\Sigma_0)} \forall_{p' \in \mathcal{C}(\Sigma_0 \cup \Sigma_1)} \forall_{l \in L_0 \cup L_{10}} tss_A \vdash \dfrac{N}{p \xrightarrow{l} p'} \Leftrightarrow tss_0 \cup tss_1 \vDash \dfrac{N}{p \xrightarrow{l} p'}$

*Proof.* The implication from left to right holds trivially since any proof structure in $tss_A$ remains valid in $tss_0 \cup tss_1$. For the implication in the other direction, all deduction rules used to deduce $tss_0 \cup tss_1 \vDash \dfrac{N}{p \xrightarrow{l} p'}$ should be in $D_0 \cup D_{10}$ since rules from $D_{11}$ have disjoint labels from both $L_0$ and $L_{10}$. We prove the claim by an induction on the depth of the proof for $tss_0 \cup tss_1 \vDash \dfrac{N}{p \xrightarrow{l} p'}$.

If the last deduction rule $d$ applied in the proof tree is in $tss_0$ and if we denote the substitution applied to $d$ by $\sigma$, then it follows from source dependency and by an induction on the source distance of the premises that the source of all premises with $\sigma$ applied to them are in $\mathcal{C}(\Sigma_0)$ and since the labels are in $L_0$, the induction hypothesis applies and all the positive premises have a proof in $tss_A$ and all the variables in their target are evaluated by $\sigma$ to terms in $\mathcal{C}(\Sigma_0)$. The variables in the target of the conclusion are source dependent and hence, the target of the conclusion with $\sigma$ applied to it is also in $\mathcal{C}(\Sigma_0)$.

If the last deduction rule $d$ applied in the proof tree is in $D_{10}$ and if we denote the substitution applied to $d$ by $\sigma$ then it follows from the hypotheses of Theorem 3.15 that one of the following two conditions should hold:

1. source of the conclusion of $d$ mentions a function symbol not in $\Sigma_0$, but then it cannot match $p$, or

2. $\rho(d, \Sigma_0)$ has a premise $t_i \xrightarrow{l_i} t'_i$ such that $l_i \notin L_0$ or $t'_i \notin \mathcal{T}(\Sigma_0, V)$. In this case, by an induction on the source distance of $t_i \xrightarrow{l_i} t'_i$, it follows that $\sigma(t_i) \in \mathcal{C}(\Sigma_0)$ and since $d \in D_{10}$, $l \in L_{10}$, thus, the induction hypothesis applies and $tss_A \vdash \dfrac{N}{\sigma(t_i) \xrightarrow{l_i} \sigma(t'_i)}$, but since $\sigma(t'_i) \notin \mathcal{T}(\Sigma_0, V)$ this transition is not provable in $tss_A$.

Hence, both cases lead to a contradiction and a rule in $D_{10}$ cannot be used in constructing a proof for $\dfrac{N}{p \xrightarrow{l} p'}$ and this concludes the proof of our claim. ⊠

For the second item, we have to prove that $\forall_{p \in \mathcal{C}(\Sigma_0)}\ \forall_{p' \in \mathcal{C}(\Sigma_0 \cup \Sigma_1)}\ \forall_{l \in L_{11}}\ tss_0 \cup tss_1 \vdash p \xrightarrow{l} p' \Leftrightarrow p = p'$. To prove this item, we can confine ourselves to rules in $D_{11}$ since rules in $D_0 \cup D_{10}$ have disjoint labels and hence cannot provide a proof for $p \xrightarrow{l} p'$. $tss_{11}$ satisfies the hypotheses of Theorem 6.46 and hence it grants $L_{11}$ transitions on $\Sigma_0$ terms. Using a similar proof as of Theorem 6.46, we can prove that for $\Sigma_0$-terms as a source, it can only prove self transitions with $L_{11}$ labels, hence, the second item. ⊠

Note that, in general, the proof obligation for orthogonality cannot be decomposed into the proof of orthogonality of two subsets of an extension. However, we conjecture that if the two subsets have sets of disjoint labels and one has a disjoint set of labels with the original TSS, the orthogonality of the combination can be guaranteed (yielding a generalization of the above theorem).

Another way to decompose orthogonality is to apply conservativity and granting extension theorems in sequence. This is possible by virtue of the following corollary which follows trivially from the definition of orthogonality (Definition 6.7).

**Corollary 6.48** Orthogonal extension is a preorder.

*Proof.* For an arbitrary $tss$, it trivially holds that $tss$ is an orthogonal extension of itself. Consider three TSS's $tss_0$, $tss_1$ and $tss_2$ with $tss_x \doteq (\Sigma_x, L_x, D_x)$ for

$x \in \{0, 1, 2\}$ such that $tss_2$ is an orthogonal extension of $tss_1$ and $tss_1$ is an orthogonal extension of $tss_0$. One can easily check that the two conditions for $tss_2$ to be an orthogonal extension of $tss_0$ hold:

1. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)}\ \forall_{l \in L_0}\ tss_2 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_1 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_0 \vDash p \xrightarrow{l} p'$, and
2. $\forall_{p,p' \in \mathcal{C}(\Sigma_0)}\ tss_2 \vDash p \sim p' \Leftrightarrow tss_1 \vDash p \sim p' \Leftrightarrow tss_0 \vDash p \sim p'$.

⊠

Using this corollary one can interleave several steps of operationally conservative and granting extensions to define different new aspects and, in the end, have an orthogonal extension of the original language. The following example illustrates applications of the above results.

**Example 6.49** (Timed-*MPA*: Orthogonality) Consider the $tss_m$ of *MPA* in Example 6.2 and $tss_t$ of Example 6.4. TSS $tss_t$ can be decomposed into the following three parts: $tss_0 \doteq (\{\underline{a}.\_, \underline{\delta}\}, A, \{\textbf{(ua)}, \textbf{(td)}\})$, $tss_1 \doteq (\{\delta, a.\_, \_ + \_\}, \{1\}, \{\textbf{(tc0)},$ $\textbf{(ta)}$, $\textbf{(d)}\})$ and $tss_2 \doteq (\{\_ + \_\}, \{1\}, \{\textbf{(tc1)}, \textbf{(tc2)}\})$.

It follows from Definition 6.42 that $tss_1$ is source preserving since:

1. the conclusions of **(ta)** and **(d)** are unifiable using any substitution, hence using the unifiers of the empty set of premises, and
2. the conclusion of **(tc0)** is unifiable using the unifiers of the premises, i.e., those that evaluate $x$ and $x'$ to the same term and $y$ and $y'$ to the same term.

It then follows from Definition 6.45 that $tss_1 \cup tss_2$ grants time transitions over *MPA* terms since

1. $tss_1$ is strictly stratified using a simple measure of size on terms, it is source preserving as shown before, and by applying unifiers to the source of conclusion of **(tc0)**, **(ta)** and **(d)**, i.e., the set $\{x + y, a.x, \delta\}$, we can cover the syntax of *MPA*,
2. in $tss_2$, deduction rules **(tc1)** and **(tc2)** have source-dependent negative premises with label 1 (note that **(tc1)** and **(tc2)** are the same as their reduced versions).

From Theorem 6.46, it follows that the extension of $tss_m$ with $tss_1 \cup tss_2$ is a granting extension, hence an orthogonal extension. Furthermore, the extension of $tss_m \cup tss_1 \cup tss_2$ with $tss_0$ is conservative, hence orthogonal, following Theorem 3.15. Since orthogonality is a preorder, we conclude that $tss_m \cup tss_t$ is an orthogonal extension of $tss_m$.

Alternatively, we could use Theorem 6.47 to obtain orthogonality by using the above results. Namely, $tss_0$ satisfies the criteria for operational conservativity of Theorem 3.15 w.r.t. $tss_m$, $tss_1 \cup tss_2$ satisfies the granting criteria of Theorem 6.46 w.r.t. $tss_m$ and finally, the labels of $tss_0$ and $tss_1 \cup tss_2$ are disjoint. Hence, using Theorem 6.47, we can conclude that $tss_m \cup tss_t$ is an orthogonal extension of $tss_m$.

To give an idea how to deal with predicates in our settings, we treat the process algebra timed-*MPA* enriched with a termination predicate and successful termination constants.

**Example 6.50** (Timed-*MPA*: Termination)

$$
\begin{array}{llll}
(1) & \textbf{(et)}\dfrac{}{\epsilon \downarrow} & \textbf{(ct0)}\dfrac{x_0 \downarrow}{x_0 + x_1 \downarrow} & \textbf{(ct0)}\dfrac{x_1 \downarrow}{x_0 + x_1 \downarrow} \\[2ex]
(2) & & \textbf{(te)}\dfrac{}{\epsilon \xrightarrow{1} \epsilon} & \textbf{(ue)}\dfrac{}{\underline{\underline{\epsilon}} \downarrow}
\end{array}
$$

Consider the above TSS which is supposed to be added to the $tss_t$ of Example 6.4. In order to deal with the termination predicate in our setting, we transform it to the form of a binary transition formula with a new label $\downarrow$. However, a choice can be made concerning the target of the newly formed formulae. In [135], it is suggested to take different fresh variables as targets of transformed premises and a new dummy constant as targets of premises. Thus, the above example would be transformed to the following TSS, where $\surd$ is the dummy constant. Using the following TSS, we can now apply Theorem 3.15 and conclude that timed-*MPA* with successful termination is an orthogonal extension of both timed-*MPA* and *MPA*.

$$
\begin{array}{llll}
(1) & \textbf{(et)}\dfrac{}{\epsilon \xrightarrow{\downarrow} \surd} & \textbf{(ct0)}\dfrac{x_0 \xrightarrow{\downarrow} y_0}{x_0 + x_1 \xrightarrow{\downarrow} \surd} & \textbf{(ct0)}\dfrac{x_1 \xrightarrow{\downarrow} y_1}{x_0 + x_1 \xrightarrow{\downarrow} \surd} \\[2ex]
(2) & & \textbf{(te)}\dfrac{}{\epsilon \xrightarrow{1} \epsilon} & \textbf{(ue)}\dfrac{}{\underline{\underline{\epsilon}} \xrightarrow{\downarrow} \surd}
\end{array}
$$

The above choice of targets for the target of transformed formulae is motivated by the desire to make the transformed TSS conform to a certain rule format, i.e., PANTH format of [135]. However, for proving orthogonality, we do not necessarily need the conformance to the PANTH format and one can take other targets for the targets. For example, a natural choice would be to take the source terms as the target and this way, we end up with the following TSS.

$$
\begin{array}{llll}
(1) & (\mathbf{et})\,\dfrac{}{\epsilon \xrightarrow{\downarrow} \epsilon} & (\mathbf{ct0})\,\dfrac{x_0 \xrightarrow{\downarrow} x_0}{x_0 + x_1 \xrightarrow{\downarrow} x_0 + x_1} & (\mathbf{ct0})\,\dfrac{x_1 \xrightarrow{\downarrow} x_1}{x_0 + x_1 \xrightarrow{\downarrow} x_0 + x_1} \\
(2) & & (\mathbf{te})\,\dfrac{}{\epsilon \xrightarrow{1} \epsilon} & (\mathbf{ue})\,\dfrac{}{\underline{\underline{\epsilon}} \xrightarrow{\downarrow} \underline{\underline{\epsilon}}}
\end{array}
$$

Actually, the above transformation may be preferable in the case of proving a granting extension since it discharges all obligation concerning unification.

### 6.4.4 Orthogonality and Equational Conservativity

The following theorems establish the link between orthogonality and equational conservativity. They are very similar to those in [135, 4] about the relation between operational and equational conservativity. The first theorem states that a sound axiomatization of an operationally conservative extension cannot induce new equalities on the old syntax.

**Theorem 6.51** (Equational Conservativity Theorem) Consider two TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$ where $tss_1$ is an orthogonal extension of $tss_0$. Also let $E_0$ be a sound and ground-complete axiomatization of $tss_0$ and $E_1$ be a sound axiomatization of $tss_1$. If $\forall_{p,p' \in \mathcal{C}}\ E_0 \vdash p = p' \Rightarrow E_1 \vdash p = p'$ then $E_1$ is an equationally conservative ground-extension of $E_0$.

*Proof.* We have to prove that $\forall_{p,p' \in \mathcal{C}(\Sigma_0)} E_0 \vdash p = p' \Leftrightarrow E_1 \vdash p = p'$. The implication from left to right is given by the hypothesis, thus it remains to prove that $E_1 \vdash p = p' \Rightarrow E_0 \vdash p = p'$. Since $E_1$ is sound, it follows that $tss_1 \vdash p \sim p'$ and since $tss_1$ is an orthogonal extension of $tss_0$, it holds that $tss_0 \vdash p \sim p'$. Then it follows from the completeness of $E_0$ that $E_0 \vdash p = p'$. ⊠

The next theorem enables us to use orthogonality as a means to equational conservativity.

**Theorem 6.52** Consider TSS's $tss_0 = (\Sigma_0, L_0, D_0)$ and $tss_1 = (\Sigma_1, L_1, D_1)$ where $tss_1$ is an orthogonal extension of $tss_0$. Also let $E_0$ and $E_1$ be sound and ground-complete axiomatizations of $tss_0$ and $tss_1$, respectively. Then, $E_1$ is an equationally conservative ground-extension of $E_0$.

*Proof.* We have to prove that $\forall_{p,p' \in \mathcal{C}(\Sigma_0)} E_0 \vdash p = p' \Leftrightarrow E_1 \vdash p = p'$. From $E_0 \vdash p = p'$, by soundness of $E_0$, $tss_0 \vdash p \sim p'$. Since $tss_1$ is an orthogonal extension of $tss_0$, we obtain $tss_1 \vdash p \sim p'$. From the hypothesis that $E_1$ is a ground-complete axiomatization of $\sim$ w.r.t. $tss_1$, we have $E_1 \vdash p = p'$. Similarly, from $E_1 \vdash p = p'$, since $E_1$ is sound, we have $tss_1 \vdash p \sim p'$. Due to orthogonality, $tss_0 \vdash p \sim p'$. The completeness of $E_0$ then gives $E_0 \vdash p = p'$. ⊠

Using the above theorem, we can obtain the equational conservativity result for timed-*MPA*.

**Example 6.53** (Timed-*MPA*: Equational Conservativity) Consider the equational theories for *MPA* and timed-*MPA* presented in Example 6.3 and 6.6. Assuming the soundness and ground-completeness of both axiomatizations which we claimed without a proof, and the orthogonality of the extension of *MPA* to timed-*MPA* which we proved in Example 6.49, we can deduce using Theorem 6.52 that the equational theory of Example 6.6 is an equationally conservative ground-extension of that of Example 6.3.

Finally, the last theorem establishes sufficient conditions for a sound equationally conservative ground-extension to be a ground-complete equational theory for the extended language. To establish such a useful result we need to eliminate the function symbols from the extending theory, as defined below.

**Definition 6.54** An equational theory $E$ on $\Sigma$ *eliminates* function symbols from $\Sigma' \subseteq \Sigma$ if and only if for all $p \in \mathcal{C}$ there exists a term $p' \in \mathcal{C}(\Sigma \setminus \Sigma')$ such that $E \vdash p = p'$.

**Theorem 6.55** (Elimination Theorem) Consider TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$ where $tss_1$ is an orthogonal extension of $tss_0$. Also let $E_0$ and $E_1$ be sound axiomatizations of $tss_0$ and $tss_1$, respectively. If $E_0$ is also ground-complete on $tss_0$, $E_1$ is an equationally conservative ground-extension of $E_0$ and $E_1$ eliminates function symbols from $\Sigma_1 \setminus \Sigma_0$, then $E_1$ is ground-complete for $tss_1$.

*Proof.* Consider two closed terms $p, p' \in \mathcal{C}(\Sigma_1)$ such that $tss_1 \vdash p \sim p'$. Since $E_1$ eliminates terms from $\Sigma_1 \setminus \Sigma_0$, there exist two terms $q, q' \in \mathcal{C}(\Sigma_0)$ such that $E_1 \vdash p = q$ and $E_1 \vdash p' = q'$. It follows from soundness of $E_1$ that $tss_1 \vdash p \sim q$ and $tss_1 \vdash p' \sim q'$ and since $\sim$ is an equivalence relation, it follows that $tss_1 \vdash q \sim q'$. But $tss_1$ is an orthogonal extension of $tss_0$ and hence, we have $tss_0 \vdash q \sim q'$. $E_0$ is a ground-complete axiomatization of $tss_0$ and thus, $E_0 \vdash q = q'$ and it follows from the equational conservativity of $E_1$ with respect to $E_0$ that $E_1 \vdash q = q'$. From $p = q$, $q = q'$ and $p' = q'$, we conclude that $E_1 \vdash p = p'$. ⊠

A typical line of reasoning starts with taking an orthogonal extension and a sound axiomatization thereof, and proving equational conservativity using Theorem 6.51. Then, by proving an elimination result for newly introduced operators, one can get completeness of the axiomatization following Theorem 6.55.

## 6.5 Conclusions

In this chapter, we defined a more relaxed notion of operational conservativity, called *orthogonality* which allows for non-destructive extension of the behavior of the old language. We gave a meta-theorem providing sufficient conditions (which are indeed more relaxed than the sufficient conditions for the traditional notion). Also, we presented the slightly more general notion of *equational conservativity for ground-extensions* and established the link between these two notions. The concepts and results were illustrated by extending the Minimal Process Algebra of [14] to a timed setting.

In [10], we design a spectrum of timed process algebras with successful termination that are related using our relaxed notion of equational conservativity. This has not been possible before (cf. [133]) due to the restrictions imposed by the old definition. In [10], we use orthogonality as a means to prove equational conservativity among these process algebras.

Extending the theory presented in this chapter with the concept of variable binding is a straightforward extension along the lines of [48]. The second enhancement of our work concerns operational extensions that require a translation of labels (using a kind of abstraction function). Finally, investigating the possibility of other realizations of orthogonality is an interesting subject for future research.

Studying extensions such as the timed extension of $\mu$CRL [109] in which the old labels are augmented with new information is also an interesting line for future work.

# Chapter 7

# Implementation

> "The construction itself is an art, its application to the world an evil parasite."
>
> [Luitzen Egbertus Jan Brouwer]

---

An earlier version of this chapter is to appear as: M.R. Mousavi, M.A. Reniers, Prototyping SOS Meta-theory in Maude, In P. Mosses and I. Ulidowski eds., *Proceedings of the 2nd Workshop on Structural Operational Semantics (SOS'05)*, Lisbon, Portugal, Electronic Notes in Theoretical Computer Science, Elsevier, July 2005.

## 7.1 Introduction

Defining a formal semantics for a language is usually among the very first steps of bringing it into the formal world. The process of defining the semantics involves many choices some of which are very implicit and hidden from the designer's naked eyes. Furthermore, there is usually no reference point to check whether the end result is "correct" and the right choices have been made during the process of defining the semantics. For a complicated language, it soon goes beyond human capabilities to keep track of the consequences of design-decisions in the semantics and one can often overlook possible counter-intuitive phenomena introduced there. Proving theorems about intuitive properties and checking several instances of system runs (according to the given semantics) against one's intuition are good ways to build insight and confidence in the semantics.

In this chapter, we report an initial attempt to implement a general-purpose tool that provides a language designer with the above possibilities for languages endowed with an Structural Operational Semantics.

As illustrated in Chapter 3, there has been a reasonable body of knowledge developed around the concept of SOS. But unfortunately, little has been done about implementing these theories. We aim at defining a framework which allows us to check the premises of some of the meta-theorems for SOS specification and further allows for animating programs according to the given semantics. The Maude term rewriting language [1] comes very handy as the base language for our implementation.

The rest of this chapter is structured as follows. In Section 7.2, we review the related work in prototyping SOS languages and checking meta-theorems about them. Afterwards, in Section 7.3, our implementation of Transition System Specifications in Maude is described. An instance of a congruence meta-theorem is then defined in Section 7.4 and implemented. Section 7.5 defines a simple operational conservativity theorem and illustrates its implementation. Section 7.6 is devoted to animating SOS specifications. Finally, Section 7.7 concludes the chapter and proposes several possible extensions of our prototype. In this chapter we recall the GSOS format (Definition 3.5) without re-stating it. The Maude code of the prototype presented in this chapter can be downloaded from the following URL: `http://www.win.tue.nl/~mousavi/sos05-meta-theory.maude`.

## 7.2 Related Work

In [99], we report our initial experiment with implementing an instance of SOS specification in the Maude rewriting logic [1] which was used as a prototype simulation and model checking environment for the particular target language. This initial prototype helped us check and remove a few "bugs" in our initial semantics.

Apart from our previous implementation, other authors have studied the, rather evident, link between the rewriting logic [80] and SOS [108] both from a theoretical [43, 80, 81, 82] as well as practical point of view [32, 33, 129, 131, 132].

In [32], the outline of a translation from Modular SOS (MSOS) [93, 94] to Maude rewriting logic is given and proven correct. The translation is quite straightforward and the main technical twist is in the decomposition of labels into the configurations in the source and the targets of the transitions which is due to the structure of labels in MSOS. The translation is fully implemented and details of this implementation can be consulted in [31]. The main difference between this research and ours stems from the fact that we take SOS as our point of departure and this may help us benefit from its theoretical history and practical popularity.

Verdejo in [129] and Verdejo and Marti-Oliet in [131, 132] report the implementation of a number of instances of SOS semantics in Maude. Our approach is very close in essence to their work in that SOS deduction rules are interpreted as Maude conditional rewrite rules. We contribute to their work by first, raising the level of abstraction a bit so that one can talk about SOS rules in general, specify and execute them and reason about them and second, we implement the slightly involved case of SOS with negative premises in our framework.

Earlier versions (of Maude) did not support conditional rewriting with rewrites as conditions. Thus, a different approach has been proposed in [80] to implement SOS, called *transitions as judgements*. In this approach each transition is implemented as a term and SOS deduction rules are implemented as rewrite rules that rewrite the transition in the conclusion to the transitions in the premises or vice versa (i.e., constructing a proof structure using a bottom-up or top-down approach). Both of these approaches have been suggested by [80] and the former has been implemented in [130]. Both transitions as rewrites and as judgements can be useful. In [131], it is reported that the transitions as rewrites approach is easier to implement and causes less complications. Furthermore, modeling transitions as rewrites allows for exploiting available search and model checking libraries implemented in Maude to investigate the behavior of a model.

LETOS [65] is a tool that generates LaTeX documents as well as executable animation code in Miranda [124] from a wide range of semantics, including some forms of SOS. A first attempt to implement an SOS meta-theorem, concerning operational conservativity of [64] is also reported in [65]. However, the implementation does not fully check this theorem and only checks the *source-dependency* requirement which is one of the hypotheses of the conservativity theorem of [64].

Centaur tool-set [28, 40] provides several formalisms for defining syntax and semantics of languages and supports them by tools for generating user-interfaces, interpreters and debuggers. Thanks to the expressive-ness and reflective semantics of Maude, all these formalisms collapse into one formalism in our implementation and so far, we do not see the need to use any other formalism or programming language to define any aspect of syntax and semantics and to implement the back-end of our tool.

## 7.3 Transition System Specifications in Maude

In this section, we formalize the concept of TSS with constant labels (Definition 3.1) in Maude. A natural extension of our implementation would be to support terms as labels.

**Formalization in Maude** Labels and variables are defined as sorts `Labels` and `Vars`, respectively. Elements of sort `Labels` are left to be defined by the user but we treat the labels as constants (possibly with some algebraic structure). Basic constructors `X- n` and `Y- n` are defined for variables $X_n$ and $Y_n$ indexed by natural numbers. A signature is to be defined per specification. Function symbols in the signature are to be defined using the Maude syntax. For example, a binary operator $\_+\_$ can be defined as `op _ + _ :  T T -> T [ctor] .`, where `T` is the given name for the sort of terms and `ctor` stands for constructor. Substitutions and matching are already defined for variables and have to be lifted by the user to the term level. We foresee the possibility of generating substitution and matching axioms automatically by examining the signature at meta-level.

Formulae $s \xrightarrow{l} s'$ and $s \stackrel{l}{\nrightarrow}$ are denoted by expressions `s == l ==> sp` and `s == l =/=>`, respectively. A TSS is a functional theory parameterized by the signature, variables and labels. However, since the parameterized modules are not supported at meta-level by the current implementation of Maude, we implement them as plain functional modules. Transforming our implementation to the parameterized setting is a matter of renaming interfaces and sort names. A deduction rule $\frac{H}{c}$ is denoted by `H === c` and deduction rules in a set are separated by commas.

Using the general implementation of TSS's and related concepts, we can specify instances of SOS specification as shown in the examples given below. Note that the examples are there for explanation purposes and do not necessarily stand for practical and meaningful instances of SOS.

**Examples** Table 7.1 shows the SOS specification of the Minimal Process Algebra (MPA) (Example 6.2) in our framework. Note that this is precisely the amount of text that has to be typed in, to benefit from the meta-theory implemented in our framework and possible extensions thereof. The Maude code is self-explanatory and is almost the same as the text appearing in Example 6.2. The signature consists of a constant `delta` for the deadlocking process, a class of unary operators `a ; _` for action prefixing with `a` being a member of the sort `BAct` (for Basic

```
fmod MPA-TSS is
 inc Term-Match .
 inc TSS-Definition .
 sort BAct .
 subsort BAct < Labels .
*** MPA Signature
 op delta   : -> T [ctor] .
 op _ ; _ : BAct T -> T [ctor] .
 op _ + _ : T T -> T [ctor] .
*** Substitutions and Matching
 op a         : -> BAct [ctor] .
 var act      : BAct .
 var sigma    : Sbst .
 vars s t sp tp  : T .
 eq sigma ( delta ) = delta .
 eq sigma (act ; s) =
    act ; sigma (s) .
 eq sigma (s + t) =
    sigma (s) + sigma (t) .
 eq vars (delta) = emptyVars .
 eq vars (act ; s) = vars (s) .
 eq vars (s + t) =
    vars (s) cup vars (t) .
 eq match (delta, delta) = emptySbst .
 eq match ((s + t), (sp + tp) ) =
    (match (s, sp), match (t, tp)) .
eq match ((act ; s), (act ; t)) =
  match (s, t) .
*** Operational Semantics of MPA
 op MPA  : -> TSS .
 eq MPA =
 ( ===
   a ; X-  0 == a ==> X-  0 ) ,
 ( X- 0 == a ==> Y-  0
   ===
   X- 0 + X- 1 == a ==> Y- 0 ) ,
 ( X- 1 == a ==> Y-  1
   ===
   X- 0 + X- 1 == a ==> Y-  1 ) .
endfm
```

**Table 7.1** Structural Operational Semantics of MPA in Maude

Actions) and a binary operator `_ + _` for nondeterministic choice. The concepts of substitution, matching and variables of a term are defined by a simple structural induction on terms (the base cases for these definitions are defined generically in the module `Term-Match`). Deduction rules define the operational semantics of action prefixing and nondeterministic choice.

Our next example is a simple extension of MPA with the aspect of timing (Example 6.4) presented in Table 7.2. In this extension, we have a new label `tick` for the time transition and a new unary operator `delay ; _` which causes a time transition to happen. Apart from the deduction rules specified before, we have to add deduction rules defining the behavior of the delay operator and also the time-deterministic nature of choice, i.e., time will only decide about non-deterministic choice if one of the two components blocks the time transition.

To simplify matters in the remainder, we assume that TSS's extending other specifications import (`include`) the theory to be extended but have all the newly introduced function symbols, labels and deduction rules in a single module.

## 7.4 A Congruence Meta-Theorem

In this section, we chose a simple congruence format, i.e., the GSOS format of [25] (see Definition 3.5) and explain its implementation in Maude.

```
fmod MPAT-TSS is
 inc MPA-TSS .
 op tick       : -> Labels [ctor] .
 op delay ; _ : T -> T [ctor] .
 eq sigma (delay ; s)  =
    delay ; sigma (s) .
 eq match ((delay ; s),
    (delay ; t)) = match (s , t) .
 eq vars (delay ; s) = vars (s) .
*** Operational Semantics of MPAT
 op MPAT  : -> TSS .
 eq MPAT =  ( MPA,
 ( ===
   delay ; X- 0 == tick ==> X- 0 ),
(( X- 0 == tick =/=> ,
   X- 1 == tick ==> Y-  1 )
   ===
  X- 0 + X- 1 == tick ==> Y-  1 ) ,
(( X- 0 == tick ==> Y- 0 ,
   X- 1 == tick =/=> )
   ===
   X- 0 + X- 1 == tick ==> Y-  0 ) ,
(( X- 0 == tick ==> Y- 0 ,
  X- 1 == tick ==> Y- 1 )
   ===
   X- 0 + X- 1 == tick ==>
                    Y-  0 + Y-  1 )) .
endfm
```

**Table 7.2** A Simple Extension of MPA with Time in Maude

**Formalization in Maude** Our formalization of the GSOS format makes use of the reflective semantics of Maude. Reflection in this context means that any rewrite theory can be interpreted as an object inside a "universal" rewrite theory. This way one can look at theories from a meta-level viewpoint and reason about them. Using this capability we examine the structure of deduction rules by first, automatically compiling a list of function symbols in the signature (with a target source `T`) using the meta-level operation `getOps` and then, checking whether the premises contain only the right kind of variables in their source and target. Again, checking the type of terms appearing in premises is performed using meta-level functions. So, our implementation remains independent from the choice of signature and the set of defined and used variables. The implemented `GSOS-Check` operator takes the name of the TSS (of type `Qid`) as a parameter, reads the signature of the TSS from the corresponding functional module, checks the conformance of rules and outputs a string which states the positive result, or alternatively, outputs one deduction rule which does not conform to the GSOS format.

**Examples** Consider the TSS's of MPAT given in Table 7.2. The following statements show how to check conformance of MPA to the GSOS format and the outcome of this check (applying a similar commands on MPA results in a similar result).

**Example** Checking the conservativity of the extension of MPA (Table 7.1) with time (Table 7.2) goes as follows.

```
Maude> red in GSOS-Check : GSOS-Chk ( 'MPAT-TSS , MPAT ) .
reduce in GSOS-Check : GSOS-Chk('MPAT-TSS,MPAT) .
rewrites: 211 in 30ms cpu (80ms real) (7033 rewrites/second)
result Message: successmsg
```

| | |
|---|---|
| `fmod Test-TSS is`<br>` inc Term-Match .`<br>` inc TSS-Definition .`<br>`*** Signature`<br>` ops a b :  -> T [ctor] .`<br>` op f _ :  T -> T [ctor] .`<br>` op l :  Labels [ctor] .` | `*** Operational Semantics`<br>`op Test :  -> TSS .`<br>`eq Test =`<br>` (       ===`<br>` f (a) == l ==> f(a)) .`<br>`endfm` |

**Table 7.3** A Simple TSS Violating GSOS Format

```
("GSOS-Check: TSS conforms to GSOS.")
```

Now, consider the TSS shown in Table 7.3. Applying GSOS-Check on this TSS results in the following error messages.

```
Maude> red in GSOS-Check : GSOS-Chk ( 'Test-TSS , Test ) .
reduce in GSOS-Check : GSOS-Chk('Test-TSS,Test) .
rewrites: 49 in 0ms cpu (0ms real) (~ rewrites/second)
result Message: errormsg(
"GSOS-Check: Error, the following rule:",
emFr === ft(a)  == l ==> ft(a),
"has more than one operator in its source of conclusion.")
```

The GSOS format only provides sufficient (and not necessary) conditions for the congruence of bisimilarity. In the above case, bisimilarity is indeed not a congruence: $a \underline{\leftrightarrow} b$ since both of them have no operational behavior but it does not hold that $f(a) \underline{\leftrightarrow} f(b)$ since the former can make a transition using the rule mentioned above while the later cannot.

## 7.5 Operational Conservativity

The following theorem is a simplification of the general theorem presented in [48].

**Theorem 7.1 (Operational Conservativity for GSOS)** Consider consistent TSS's $tss_0 = (\Sigma_0, V, L_0, D_0)$ and $tss_1 = (\Sigma_1, V, L_1, D_1)$ in the GSOS format. $tss_0 \cup tss_1$ is an operationally conservative extension of $tss_0$ if for each deduction rule $d \in D_1$, one of the following conditions hold:

1. $d$ mentions a function symbol from $L_1 \setminus L_0$ in the source of its conclusion, or
2. $d$ has a positive premise $x_i \xrightarrow{l_{ij}} y_i$ with $l_{ij} \in L_1 \setminus L_0$.

**Formalization in Maude** Formalization of the conservativity meta-theorem goes along the same lines as that of congruence meta-theorem. First, we compile a list of function symbols and labels in the extended and extending TSS's and then check the deduction rules of the extending TSS to either include a fresh function symbol in the source of conclusion or a fresh label in the positive premises.

**Example** Checking the conservativity of the extension of MPA (Table 7.1) with time (Table 7.2) goes as follows.

```
Maude> red in CONSV-Check : Cons-Chk ( 'MPA-TSS , MPA, 'MPAT-TSS, MPAT ) .
reduce in CONSV-Check : Cons-Chk('MPA-TSS,MPA,'MPAT-TSS,MPAT) .
rewrites: 14 in 0ms cpu (0ms real) (~ rewrites/second)
result Message: successmsg
("CONS-Check: Operational conservativity theorem checked successfully.")
```

Trying the same routine on a non-conservative extension results in an error message which points out the deduction rule and the hypotheses of the conservativity theorem that has been violated.

## 7.6 Animating SOS

**Motivation** Despite their operational nature, SOS specifications are not in general executable. As shown in [25], slight extensions to GSOS easily ruin the decidability of proving a transition. To add to the complications, it was shown in [61] that not all SOS specifications are meaningful, in that they may not define a transition relation at all or they may ambiguously allow for more than one transition relation. By taking GSOS as a framework, one may be relieved of these hassles. Our animation method does not require the TSS to be in GSOS. However, it guarantees to terminate and produce a sound result if the TSS is in a superset of GSOS specifications called *strictly and finitely stratified* TSS's. Next, we precisely define what does it mean for a transition to be provable from a TSS and how this concept is formalized in Maude.

**Definition 7.2 (Finite Stratification)** A stratification measure is *finite* if its range is the natural numbers. A transition system specification is called *finitely stratified* if and only if there exists a finite stratification function for it.

**Proposition 7.3** A TSS in GSOS is strictly and finitely stratified.

**Formalization in Maude** We interpret deduction rules as conditional rewrite rules. In order to check for possible transitions for a closed term $s$, we first look for a deduction rule $d \in tss$ of the form $\frac{H}{s' \xrightarrow{l} t}$ such that $s'$ can match (i.e., is unifiable with) $s$. The unification of $s'$ with $s$ results in a substitution $\sigma_0$ evaluating the variables of $s'$. We aim at completing $\sigma_0$ into $\sigma$ such that first, $\sigma$ evaluates all the variables in $d$ (thus, the variables in $t$), second, all positive premises evaluated by $\sigma$ are provable from $tss$ and finally, negative premises evaluated by $\sigma$ cannot be contradicted by a proof from $tss$. To this end, we examine the premises in the following order.

We search for premises of $d$ of which its source is evaluated by the substitution $\sigma_j$ constructed so far.[1] If the premise is a negative one, we make sure that this fully evaluated premise cannot be contradicted by a proof from $tss$. If it is a positive premise of the form $t_i \xrightarrow{l_i} t'_i$, we try to construct a proof for a transition of $\sigma_j(t_i)$ to evaluate the variable in $t'_i$. If we succeed in constructing such a proof, we add the valuation of the variable in $t'_i$ to $\sigma_j$ resulting in $\sigma_{j+1}$. This process continues until no premise remains to be examined.

Each of the above mentioned steps is implemented as a conditional rewrite rule, rewriting a set of premises and a partial substitution to a (possibly more complete) substitution. The transition of term $s$ is then modeled as a conditional rewrite rule from $\sigma_0(s)$ to $\sigma_n(t)$ where $\sigma_n$ results from the rewrite rules of the procedure described above. For pure TSS's [64] such a substitution evaluates all variables in $t$ (the target of the transition). For non-pure TSS's, variables in $t$ that are not evaluated by the above procedure are mapped in $\sigma_n$ to an arbitrary closed term (again using a rewrite theory).

Next, we quote an excerpt of the Maude code implementing this procedure.

```
crl ( tss  |- ( s == l ==> ) ) =>
    ( ( sigma, rho ) ( target ( conc ( rule ))))
if
  ( rule , tssp )  := tss                        /\
  sigma := match (  conc(rule), ( s == l ==> ))   /\
  ( tss ||- ( sigma (prem(rule)))) =>  rho       /\
  ( ( sigma , rho ) :: Sbst )                    /\
  ( closed ( (sigma, rho) (target (conc(rule) ))))  .
```

In the above code, `crl` stands for conditional rewriting which rewrites the term before the arrow `=>` to the term after provided the condition specified by the `if` clause holds. In this case, the term before the rewrite arrow consists of the TSS under consideration (`tss`) the source (`s`) and the label (`l`) of the transition. The term after the rewriting arrow is the target of the conclusion of the matching deduction rule (`rule`) with the substitution (`sigma, rho`) to be constructed by the above mentioned procedure applied to it. In the condition part of the rewrite

[1] In a large class of TSS's such a premise can be found. Such TSS's are theoretically characterized as *pure and well-founded* TSS's [64]. For TSS's that do not have such property, a premise is chosen arbitrarily and different closed substitutions for its source are examined.

rule, first, pattern matching is used to pick an arbitrary rule from the TSS. Then, it is checked whether there is a substitution `sigma` matching the source of the rule and `s`. Next, it is checked whether a substitution `rho` can be constructed so that the premises of the rule can be satisfied (to be explained further in the remainder). If such a substitution can be found and it evaluates all variables in the target of the conclusion of `rule`, then the animation procedure has reached its goal. Otherwise, if all the premises are satisfied and still some variables in the target of the conclusion remain to be evaluated, they can be chosen non-deterministically from the set of closed terms. Here, we omit the case (of non-pure thus, non-GSOS rules) where the resulting substitution does not evaluate all variables in the target of the conclusion of `rule`.

We distinguish the following two cases for checking the premises of the deduction rule. If the premise is a positive one, then the check is nothing but looking for a transition from the source which matches the target of the premise. The matching substitution `sigma` is then used to evaluate the rest of the premises.

```
crl ( tss ||- ( s == l ==> t ) )  => sigma
if
  ( tss |- ( s == l ==> ) )  => sp /\
  sigma := match ( t, sp )           .
```

If the premise is a negative one, we use the meta-level method `metaRewrite` to check whether a contradicting rewrite (transition) can be found using the same rewriting theory. Note that the check for negative premises does not add any information with respect to the substitution under construction. Thus, the result of the rewrite is the empty substitution (`emSbst`). Again in both of these cases, we omit the rewrite rules dedicated to the cases where the chain of premises is broken (i.e., the rule is not pure) and no transition with a closed source can be found among the evaluated premises.

```
crl ( tss ||- (s == l =/=> ) )  => ( emSbst  )
if
  possible? := metaRewrite( ['TSS-Animation],
                            upTerm( ( tss |- ( s == l ==> ) ) ), 1 )  /\
  (possible? :: ResultPair )                       .
```

The above procedure, upon termination, gives us a complete proof for the transition with a guarantee that negative premises cannot be refuted using our rewrite theory, thus, using the SOS semantics. However, in general this procedure need not terminate. Consider the following two SOS specifications.

```
a == l ==> a             a == l =/=>
===                      ===
a == l ==> a             a == l ==> a
```

The Maude tool crashes when trying to animate any of the above two TSS's since the procedure results in an infinite chain of rewrites each being a condition for the next. However, this problem does not occur in GSOS specifications and in general, strictly and finitely stratified TSS's because for such TSS's checking conditions of each rewrite results in a condition with a lower stratification measure. Hence, the depth of conditional rewrite checks for a transition is always finite. Also, breadth of this search is always finite, since we can only specify a finite number of rules each having a finite number of premises. If the proof search is successful on all premises, it provides us with a substitution that evaluates the variables in the target of the deduction rule and hence, we are able to find a possible transition for term $s$ using the label and evaluated target of the conclusion of the deduction rule.

It is worth mentioning that this procedure is non-deterministic in that there may be several provable transitions for a closed term $s$. The Maude semantics has an inherent support for non-deteriministic rewrite theories and hence the choice among such transitions remains non-deterministic and is eventually made by the Maude rewriting engine. Using the Maude tool one can browse through provable transitions, check for provability of a particular transition and even use logical formulae (Linear Temporal Logic (LTL) formulae) to model check properties of transitions and runs.

**Example** Consider the TSS MPA of Table 7.1. We can animate a transition for the term `a ; (( a ; delta) + delta )` as follows.

```
Maude> rew in TSS-Animation :
  ( MPA |- ( a ; ( a ; delta + delta ) == a ==> ) ) .
rewrite in TSS-Animation : MPA |- a ; (a ; delta + delta) == a ==> .
rewrites: 13 in 0ms cpu (0ms real) (~ rewrites/second)
result T: a ; delta + delta
```

## 7.7 Conclusions and Future Extensions

In this paper, we presented an initial attempt to implement SOS meta-theory in Maude. Our implementation defines a basic SOS framework with constant labels and provides a way to prove congruence and operational conservativity meta-theorems. Furthermore, it allows for animating SOS specifications.

Maude was a very convenient choice for our implementation. In particular, the correspondence between rewrites and transitions simplified the translation from SOS to Maude. The reflective semantics of Maude was crucial in our implementation. We expect easier and more efficient implementations as the meta-level facilities provided by Maude improve gradually.

In order to turn this prototype into a full-fledged tool for SOS, we foresee the following possible extensions:

1. Implementing the more general SOS frameworks and their corresponding meta-theorems: There are more general SOS frameworks that allow for terms as labels, multi-sorted and binding signatures. Implementing such frameworks increases the applicability of our tool. Furthermore, the meta-theorems we implemented in this paper are among the simplest versions of the available meta-theorems for congruence and operational conservativity. By extending the SOS framework to more general settings, implementing more general meta-theorems such as those of [96, 48] would be beneficial;

2. Generating sound and (ground-)complete equational theories: A class of meta-theorems that we did not address in this paper concerns generating equational theories from SOS specifications (see [3], for example). These meta-theorems also have an algorithmic nature and can be implemented in our framework;

3. Generating natural language documentation (and possibly research papers!) from the specified semantics;

4. Automatically generating the term matching and substitution definitions: To check the congruence and operational conservativity meta-theorems, we used routines that extract function symbol definitions from a theory. Using similar routines we may automate the substitution and matching procedures and make the Maude code for SOS specifications even more compact;

5. Building a graphical user interface and importing SOS specifications from a general (e.g., XML) input format.

# Chapter 8

# SOS with Data

> "Errors using inadequate data are much less than those using no data at all."
>
> [Charles Babbage]

A summarized version of this chapter has appeared as: M.R. Mousavi, M.A. Reniers, J.F. Groote, Congruence for SOS with Data, In P. Panangaden ed., *Proceedings of Nineteenth Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, Turku, Finland, pp. 303-312, IEEE Computer Society Press, July 2004. A detailed version has appeared as: M.R. Mousavi, M. Reniers, J. F. Groote, Notions of Bisimulation and Congruence Formats for SOS with Data, *Information and Computation*, 200(1):107–147, Elsevier Science B.V., 2005.

## 8.1 Introduction

From early beginnings, SOS has been used for languages with data as an integral part of their operational state (e.g., the original report on SOS contains several examples of state-bearing transition system specifications [106]). Programming languages have traditionally had a notion of data. As systems get more complex, the integration of a data state in their semantics becomes more vital. Besides the systems that have an explicit notion of data such as [17] and [35], real-time languages [9, 29, 62, 71] and hybrid languages [41, 20] are other typical examples of systems in which a data state shows itself in the operational semantics in one way or another. However, the introduction of data turns out not to be as trivial as it seems and leads to new semantical issues such as adapted notions of bisimilarity [29, 62, 41, 95].

To the best of our knowledge, no standard congruence format for these different notions of bisimilarity with a data state has been proposed so far. Hence, most of the congruence proofs are done manually [95] or are just neglected by making a reference to a standard format that does not cover the data state [29]. The proposal that comes closest ([26]) is unfinished and encodes rules for state-bearing processes into rules without a state, for which a format is given.

In this chapter, we address the implications of the presence of a data state on notions of bisimilarity and propose standard formats that induce congruence with respect to these notions of bisimilarity.

The rest of this chapter is structured as follows. In Section 8.2, we set the scene by extending our SOS framework to the setting with data and by defining notions of bisimilarity in this new setting. In this section, we also sketch the relationship between these notions and point out their application areas. The main contribution of this chapter is introduced in Section 8.3, where we define standard syntactic formats for proving congruence with respect to the defined notions of bisimilarity. Furthermore, we give a full comparison between congruence results for the notions of bisimilarity with data. Subsequently, Section 8.4 presents applications of the proposed theory on transition system specifications from the literature in the domains of coordination languages, real-time process algebra, and hybrid process algebra. Finally, Section 8.5 concludes the chapter and presents possible extensions of the proposed approach.

## 8.2 Preliminaries

### 8.2.1 Basic Definitions

We assume infinite and disjoint sets of process variables $V_{\mathrm{p}}$ (typical members: $x, y, x_i, y_i \ldots$) and data variables $V_{\mathrm{d}}$ (typical member: $v$). A process signature $\Sigma_{\mathrm{p}}$

is a set of pairs $(f, n)$ where $f$ is a function symbol and $n$ is its fixed arity (denoted by $ar(f)$ in the rest of the chapter). Functions with zero arity are called constants (typical members: $a$, $b$, $c$). Process terms $t \in \mathcal{T}(\Sigma_{\mathrm{p}}, V_{\mathrm{p}})$ are defined inductively as expected (see Definition 2.2) Process terms are typically denoted by $t, t', t_i, \ldots$. Similarly, data terms $u \in \mathcal{T}(\Sigma_{\mathrm{d}}, V_{\mathrm{d}})$ are defined inductively based on a data signature $\Sigma_{\mathrm{d}}$ and the set of variables $V_{\mathrm{d}}$ and typically denoted by $u, u', u_i, u'_i, \ldots$. Closed terms $\mathcal{C}(\Sigma_{\mathrm{p}})$ and $\mathcal{C}(\Sigma_{\mathrm{d}})$ in each of these contexts are defined as expected (closed process terms are typically denoted by $p, q, p', q', p_i, q_i, p'_i, q'_i \ldots$).

A process substitution ($\sigma$ or $\sigma'$) replaces a process variable in an open process term with another (possibly open) process term. A data substitution ($\xi$) replaces a data variable in an open data term with another (possibly open) data term. The set of variables appearing in term $t$ is denoted by $vars(t)$.

**Definition 8.1 (Transition System Specification with Data)** A *transition system specification with data* is a tuple $(\Sigma_{\mathrm{p}}, \Sigma_{\mathrm{d}}, V_p, V_d, L, Rel, D)$ where $\Sigma_{\mathrm{p}}$ is a process signature, $\Sigma_{\mathrm{d}}$ is a data signature, $V_p$ and $V_d$ are sets of process and data variables, $L$ is a set of labels (with typical members $l, l', l_0, \ldots$), $Rel$ is a set of (unary) relation symbols and $D$ is a set of deduction rules. For all $r \in Rel$, $l \in L$ and $s, s' \in \mathcal{T}(\Sigma_{\mathrm{p}}, V_p) \times \mathcal{T}(\Sigma_{\mathrm{d}}, V_d)$ we define that $(s, l, s') \in r$ is a *formula*. A deduction rule $dr \in D$ is defined as a pair $(H, c)$ where $H$ is a set of formulae and $c$ is a formula. The formula $c$ is called the *conclusion* and the formulae from $H$ are called *premises*.

Notions of open and closed and the concept of substitution are lifted to formulae in the natural way. As usual, we denote formula $(s, l, s') \in r$ by $s \xrightarrow{l}_r s'$ and deduction rules $(H, c)$ by $\dfrac{H}{c}$.

Using the re-defined notion of deduction rule, the notion of *proof* of a formula (Definition 2.5) carries over naturally to the setting with data.

Note that in this paper, we only consider transition relations and notions of bisimulation on closed terms. The techniques developed in [110] can be used to define these concepts on open terms.

### 8.2.2 Notions of Bisimilarity

The introduction of data to the state adds a new dimension to the notion of bisimilarity. One might think that we can easily deal with data states by imposing the original notion of strong bisimilarity [86, 102] to the extended state. In such a case processes are compared regardless of the data they have. Our survey of the literature has revealed that such a notion of strong bisimilarity is not used at all. In this article, we restrict ourselves to comparing processes with respect to the same data state. In this way, we get to what we call a *state-based bisimilarity*, depicted in Figure 8.1.

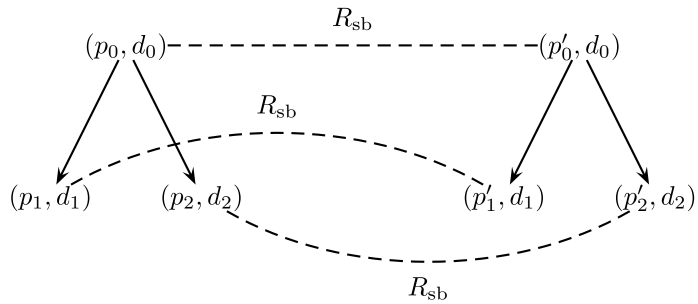


**Figure 8.1** State-based Bisimilarity

**Definition 8.2 (State-based Bisimilarity)** A relation $R_{\mathrm{sb}} \subseteq (C(\Sigma_{\mathrm{p}}) \times C(\Sigma_{\mathrm{d}})) \times (C(\Sigma_{\mathrm{p}}) \times C(\Sigma_{\mathrm{d}}))$ is a *state-based bisimulation* relation if and only if $\forall_{p,q,d_0,d_1,r,l}\ (p, d_0)\ R_{\mathrm{sb}}\ (q, d_1)) \Rightarrow d_0 = d_1 \wedge$

1. $\forall_{p',d'}\ (p, d_0) \xrightarrow{l}_r (p', d') \Rightarrow \exists_{q'}\ (q, d_0) \xrightarrow{l}_r (q', d')\ \wedge (p', d')\ R_{\mathrm{sb}}\ (q', d')$;
2. $\forall_{q',d'}\ (q, d_1) \xrightarrow{l}_r (q', d') \Rightarrow \exists_{p'}\ (p, d_1) \xrightarrow{l}_r (p', d')\ \wedge (p', d')\ R_{\mathrm{sb}}\ (q', d')$.

Two closed state terms $(p, d)$ and $(q, d)$ are *state-based bisimilar*, denoted by $(p, d) \underline{\leftrightarrow}_{sb} (q, d)$, if and only if there exists a state-based bisimulation relation $R_{\mathrm{sb}}$ such that $(p, d)\ R_{\mathrm{sb}}\ (q, d))$.

**Definition 8.3 (Process-congruence)** An equivalence relation $\sim \subseteq (C(\Sigma_{\mathrm{p}}) \times C(\Sigma_{\mathrm{d}})) \times (C(\Sigma_{\mathrm{p}}) \times C(\Sigma_{\mathrm{d}}))$ is called a *process-congruence* w.r.t. $(f, ar(f)) \in \Sigma_{\mathrm{p}}$ if and only if for all $\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1} \in \mathcal{C}(\Sigma_{\mathrm{p}})$, for all $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$, if $(p_i, d) \sim (q_i, d)$ (for $0 \leq i < ar(f)$), then $(f(\overrightarrow{p}_{ar(f)-1}), d) \sim (f(\overrightarrow{q}_{ar(f)-1}), d)$. Furthermore, $\sim$ is called a *process-congruence* for a transition system specification if and only if it is a process-congruence w.r.t. all members of the process signature.

**Example 8.4** Consider a transition system specification, where the signature consists of three (distinct) process constants $a$, $b$ and $c$, one binary process function $f$, and three (distinct) data constants $d$, $d'$ and $d''$, and the deduction rules are the following.

$$(1)\ \frac{}{(a, d) \xrightarrow{l} (a, d'')} \qquad (2)\ \frac{}{(a, d') \xrightarrow{l} (a, d')} \qquad (3)\ \frac{}{(b, d) \xrightarrow{l} (b, d'')}$$

$$(4)\ \frac{}{(c, d) \xrightarrow{l} (a, d'')} \qquad (5)\ \frac{(x_0, v) \xrightarrow{l} (y, v')}{(f(x_0, x_1), v) \xrightarrow{l} (x_1, d')}$$

Then, the following state-based bisimilarities hold:

$$(a,d)\ \underline{\leftrightarrow}_{sb}\ (b,d), \qquad (b,d)\ \underline{\leftrightarrow}_{sb}\ (c,d).$$

However, the following state-based bisimilarities do not hold: $(a,d')\ \underline{\leftrightarrow}_{sb}\ (b,d')$ and $(f(c,a),d)\ \underline{\leftrightarrow}_{sb}\ (f(c,b),d)$. From the latter case, we can observe that state-based bisimilarity is not a process-congruence for the above transition system specification.

State-based bisimilarity is a rather weak notion of bisimilarity for most practical examples. The problem lies in the fact that in this notion of bisimilarity the process parts are only related with respect to a particular data state. Thus, if the common initial data state is not known (e.g., if the components have to start their execution on the result of an unknown or non-deterministic process), then state-based bisimilarity is not useful.

This problem leads to the introduction of a new notion of bisimilarity which takes all possible initial states into account [63, 62]. We call this notion *initially stateless bisimilarity* (see Figure 8.2). This notion of bisimilarity is very useful for the case where components are composed sequentially. In such cases, when we prove that two components are bisimilar, we do not rely on the initial starting state and thus, we allow for sequential composition with any other component.

**Definition 8.5 (Initially Stateless Bisimilarity)** Two closed process terms $p$ and $q$ are *initially stateless bisimilar*, denoted by $p\ \underline{\leftrightarrow}_{isl}\ q$, if and only if there exists a state-based bisimulation relation $R_{\mathrm{sb}}$ such that $(p,d)\ R_{\mathrm{sb}}\ (q,d)$ for all $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$.

For initially stateless bisimilarity (and also for stateless bisimilarity, to be defined shortly), congruence is defined as expected (see Definition 3.3).

**Example 8.6** Consider the transition system specification of Example 8.4. The following initially stateless bisimilarities hold $b\ \underline{\leftrightarrow}_{isl}\ c$ and $f(b,c)\ \ \underline{\leftrightarrow}_{isl}\ \ f(c,c)$ but the following initially stateless bisimilarities do not hold $a\ \underline{\leftrightarrow}_{isl}\ b$ and $f(c,a)\ \ \underline{\leftrightarrow}_{isl}\ \ f(c,b)$. We observe that the previous problem of congruence does not exist anymore for initially stateless bisimilarity. Later on, in Example 8.36, we show that for this transition system specification, initially stateless bisimilarity is indeed a congruence.

However, initially stateless bisimilarity does not solve all problems, either. If there is a possibility of change in the intermediate data states (by an outside process), then initially stateless bisimilarity is not preserved in such an environment. This, for instance, happens in open concurrent systems.

Stateless bisimilarity [29, 41, 63, 95], shown in Figure 8.3, is the solution to this problem and the finest notion of bisimilarity for state-bearing processes that one

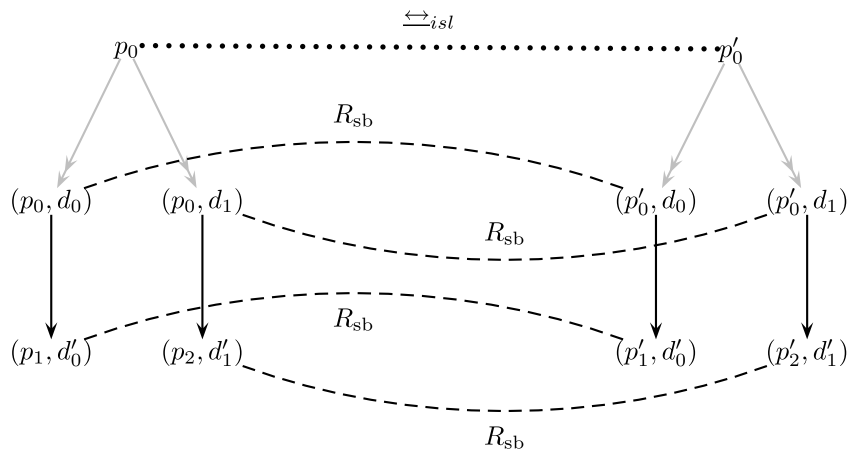


**Figure 8.2** Initially Stateless Bisimilarity

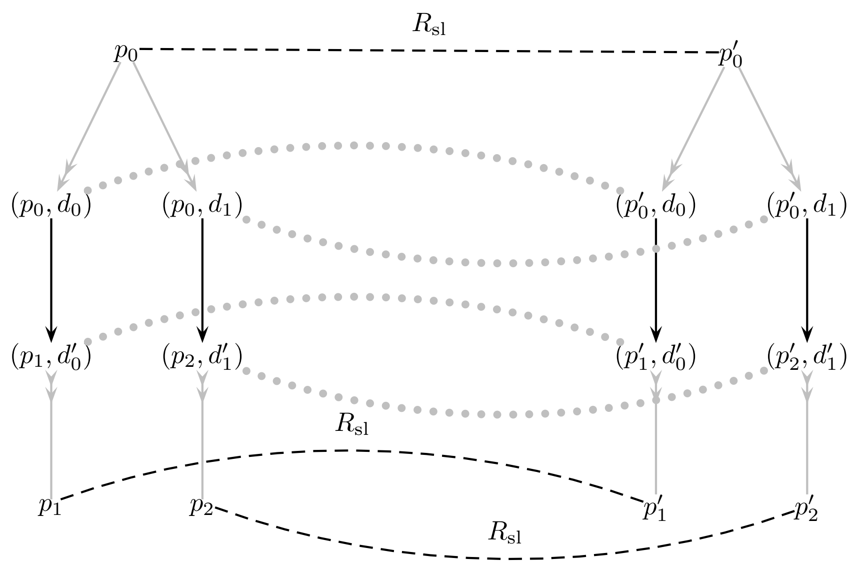


**Figure 8.3** Stateless Bisimilarity

can find in the literature. Two process terms are stateless bisimilar if, for all identical data states, they can mimic transitions of each other and the resulting process terms are again stateless bisimilar. In other words, we compare process terms for all identical data states and allow all sorts of change (interference) in the data part after each transition.

**Definition 8.7 (Stateless Bisimilarity)** A relation $R_{\mathrm{sl}} \subseteq C(\Sigma_{\mathrm{p}}) \times C(\Sigma_{\mathrm{p}})$ is a *stateless bisimulation* relation if and only if $\forall_{p,q,d,r,l}\ p\ R_{\mathrm{sl}}\ q \Rightarrow$

1. $\forall_{p',d'}\ (p,d) \xrightarrow{l}_r (p',d') \Rightarrow \exists_{q'}\ (q,d) \xrightarrow{l}_r (q',d') \wedge p'\ R_{\mathrm{sl}}\ q'$;
2. $\forall_{q',d'}\ (q,d) \xrightarrow{l}_r (q',d') \Rightarrow \exists_{p'}\ (p,d) \xrightarrow{l}_r (p',d') \wedge p'\ R_{\mathrm{sl}}\ q'$.

Two closed process terms $p$ and $q$ are *stateless bisimilar*, denoted by $p \underline{\leftrightarrow}_{sl}\ q$, if and only if there exists a stateless bisimulation relation $R_{\mathrm{sl}}$ such that $p\ R_{\mathrm{sl}} q$.

**Example 8.8** Consider the transition system specification of Example 8.4. None of the non-trivial examples of bisimilarity hold anymore for stateless bisimilarity. Namely, it does not hold that $a \underline{\leftrightarrow}_{sl}\ b$, $a \underline{\leftrightarrow}_{sl}\ c$ or $b \underline{\leftrightarrow}_{sl}\ c$. From these one may conclude that stateless bisimilarity is a congruence for the above transition system specification. We prove this formally in Example 8.16.

None of the three notions of bisimilarity is the perfect notion. State-based bisimilarity is the easiest one to check and establish but is not very robust in applications. It is most suitable for systems that are not subject to any further composition and interference. Initially stateless bisimilarity is a bit more difficult to check and establish but is more robust and suitable for systems that are amenable to further sequential compositions. Finally, stateless bisimilarity is the hardest one to establish but it is considered the most robust one for open concurrent systems. In general, a compromise has to be made in order to find the right level of robustness and strength and as a result the most suitable notion of bisimilarity has to be determined for each language/application separately.

A common practice in establishing bisimulation relations for concurrent systems is to transform them to nondeterministic sequential systems preserving stateless bisimilarity and then using initially stateless bisimilarity in that setting [63]. Another option for open systems with a limited possibility of intervention from the environment is to parameterize the notion of bisimilarity with an interference relation [63, 38, 41]. Our congruence format for state-based bisimilarity can be adapted to the parameterized notion of bisimilarity.

Next, we compare the above three notions of bisimilarity.

### 8.2.3 Comparing the Notions of Bisimilarity

In Examples 8.4 and 8.8, we have shown that two processes $b$ and $c$ are state-based bisimilar (w.r.t. data state $d$) but stateless bisimilarity fails to hold between them. Thus, we may infer that stateless bisimilarity is finer than state-based bisimilarity (w.r.t. a particular data state). The following corollary states that if a state-based bisimulation relation is closed under the change of data state then it induces a stateless bisimilarity.

**Corollary 8.9** Let $R$ be a state-based bisimulation relation. If $\forall_{p,q}$ $(\exists_d$ $(p,d)$ $R$ $(q,d) \Rightarrow \forall_{d'}$ $(p,d')$ $R$ $(q,d')$ then $\forall_{p,q}$ $(\exists_d$ $(p,d)$ $R$ $(q,d) \Rightarrow p \leftrightarroweq_{sl} q)$.

Finally, the following corollary positions initially stateless bisimilarity in between the two other notions of bisimilarity we have discussed so far.

**Corollary 8.10** For two arbitrary closed process terms $p$ and $q$, we have

1. if $p \leftrightarroweq_{sl} q$, then $p \leftrightarroweq_{isl} q$;
2. $p \leftrightarroweq_{isl} q$ if and only if, $(p,d) \leftrightarroweq_{sb} (q,d)$ for all closed data terms $d$.

Again, in Examples 8.4 and 8.6, we have shown that $a$ and $b$ are state-based bisimilar with respect to $d$ but they are not initially stateless bisimilar. Thus, state-based bisimilarity (with respect to a particular data state) is strictly weaker than initially stateless bisimilarity.

The following corollary states that stateless bisimilarity implies state-based bisimilarity with respect to all data states.

**Corollary 8.11** For two arbitrary closed process terms $p$ and $q$: if $p \leftrightarroweq_{sl} q$, then $(p,d) \leftrightarroweq_{sb} (q,d)$ for all $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$.

## 8.3 Standard Formats for Congruence

In this section, we present standard formats and prove congruence results with respect to aforementioned notions of bisimilarity. To do this, we extend the tyft format of [62] with data in three steps for stateless, state-based, and initially stateless bisimilarity. Finally, we present how our formats can be extended to cover tyxt rules and rules containing predicates and negative premises (thus, extending the PANTH format [135] with data).

### 8.3.1 Congruence Format for Stateless Bisimilarity

In this article, we allow for deduction rules that adhere to the tyft-format with respect to the process terms and are not restricted in the data terms. This format is called process-tyft.

**Definition 8.12 (Process-tyft)** Let $(\Sigma_{\mathrm{p}}, \Sigma_{\mathrm{d}}, V_{\mathrm{p}}, V_{\mathrm{d}}, L, D(Rel))$ be a transition system specification. A deduction rule $dr \in D(Rel)$ is in process-tyft format if it is of the form

$$(dr)\ \ \frac{\{(t_i,u_i) \xrightarrow{l_i}_{r_i} (y_i,u_i') \mid i \in I\}}{(f(\overrightarrow{x}_{ar(f)-1}),u) \xrightarrow{l}_{r} (t',u')},$$

where $I$ is a set of indices, $r \in Rel$, $l \in L$, $(f, ar(f)) \in \Sigma_\mathrm{p}$, $t' \in \mathcal{T}(\Sigma_\mathrm{p}, V_\mathrm{p})$, $u, u' \in \mathcal{T}(\Sigma_\mathrm{d}, V_d)$, the variables $x_0, \ldots, x_{ar(f)-1}$ and $y_i$ $(i \in I)$ are all distinct variables from $V_\mathrm{p}$, and, for all $i \in I$: $\rightarrow_{r_i} \in Rel$, $l_i \in L$, $t_i \in \mathcal{T}(\Sigma_\mathrm{p}, V_\mathrm{p})$ and $u_i, u_i' \in \mathcal{T}(\Sigma_\mathrm{d}, V_\mathrm{d})$.

We name the set of process variables appearing in the source of the conclusion $X_\mathrm{p}$ and in the target of the premises $Y_\mathrm{p}$. The two sets $X_\mathrm{p}$ and $Y_\mathrm{p}$ are obviously disjoint following the requirements of the format. The above deduction rule is called an *$f$-defining deduction rule*.

A transition system specification is in process-tyft if all its deduction rules are in the process-tyft format.

It turns out that for any transition system specification in the process-tyft format, stateless bisimilarity is a congruence. For simplicity in proofs, we require the acyclicity of the variable dependency graph (a slight variation of Definition 3.2), as well. However, this requirement can be removed using the result of [46].

**Theorem 8.13** If a transition system specification is in the process-tyft format, then stateless bisimilarity is a congruence for that transition system specification.

Before we prove this theorem, we first define the closure of a relation under stateless congruence and give and prove a lemma that is very useful in the proof of Theorem 8.13.

**Definition 8.14 (Closure under stateless congruence)** Let $R \subseteq C(\Sigma_\mathrm{p}) \times C(\Sigma_\mathrm{p})$. We define the relation $\tilde{R} \subseteq C(\Sigma_\mathrm{p}) \times C(\Sigma_\mathrm{p})$ to be the smallest congruence on $C(\Sigma_\mathrm{p})$ such that the relation $R$ is contained in $\tilde{R}$. Formally, $\tilde{R}$ is defined to be the smallest relation that satisfies:

1. $\tilde{R}$ is reflexive;
2. $R \subseteq \tilde{R}$;
3. $f(\overrightarrow{p}_{ar(f)-1})\ R\ f(\overrightarrow{q}_{ar(f)-1}))$ for all $(f, ar(f)) \in \Sigma_\mathrm{p}$,
   and for all $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1} \in \mathcal{C}(\Sigma_\mathrm{p})$ such that $\overrightarrow{p}_{ar(f)-1}\ R\ \overrightarrow{q}_{ar(f)-1}$.

**Lemma 8.15** Let $R \subseteq \mathcal{C}(\Sigma_\mathrm{p}) \times \mathcal{C}(\Sigma_\mathrm{p})$ and $t \in \mathcal{T}(\Sigma_\mathrm{p}, V_\mathrm{p})$. For any two process substitutions $\sigma$ and $\sigma'$ such that $\sigma(x)\ \tilde{R}\ \sigma'(x)$ for all $x \in vars(t)$, we have $\sigma(t)\ R\ \sigma'(t)$.

*Proof.* By induction on the structure of process term $t$. See [64] for a complete proof. ⊠

*Proof of Theorem 8.13* It suffices to prove that stateless bisimilarity is a congruence for each of the process functions of $\Sigma_{\mathrm{p}}$. Let $(f, ar(f)) \in \Sigma_{\mathrm{p}}$, $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1} \in \mathcal{C}(\Sigma_{\mathrm{p}})$ and suppose that $\overrightarrow{p}_{ar(f)-1} \underline{\leftrightarrow}_{sl} \overrightarrow{q}_{ar(f)-1}$. This means that there are stateless bisimulation relations $R_i$ (for $0 \leq i < ar(f)$) that witness these stateless bisimilarities. Let $R$ be the union of these relations $R_i$: $R = \bigcup_{i=0}^{ar(f)-1} R_i$. Obviously $R$ is also a stateless bisimulation relation. We prove that the relation $\tilde{R}$ contains the pair $(f(\overrightarrow{p}_{ar(f)-1}), f(\overrightarrow{q}_{ar(f)-1}))$ and that it is a stateless bisimulation relation. The first claim is obvious from the definition of $\tilde{R}$.

So, we only have to prove the following for any $p \; \tilde{R} \; q$: if for arbitrary $r$, $l$, $p'$, $d$ and $d'$, $(p, d) \xrightarrow{l}_r (p', d')$, then there exists a $q'$ such that $(q, d) \xrightarrow{l}_r (q', d')$ and $p'$ $\tilde{R}$ $q'$ and vice versa for transitions of $q$. Due to symmetry, it suffices to provide the proofs for the transitions of $p$ only.

We prove this by induction on the depth of the proof of a transition. We do not show the proof for the induction base as it is an instance of the proof of the induction step where there are no premises.

For the induction step, we distinguish three cases based on the structure of the definition of $\tilde{R}$. In case the pair $(p, q)$ is contained in $\tilde{R}$ due to reflexivity of $\tilde{R}$ or due to the requirement that $\tilde{R}$ contains $R$, the proof is obvious (and requires no induction at all). For the remaining case, we find $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(\overrightarrow{q}_{ar(f)-1})$ for some $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1}$ such that $\overrightarrow{p}_{ar(f)-1} \tilde{R} \overrightarrow{q}_{ar(f)-1}$. The last step of the proof of the transition of $p$ is due to the application of a deduction rule of the following form:

$$\frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') \mid i \in I\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) \xrightarrow{l}_r (t', u')}.$$

This means that there are a process substitution $\sigma$ and a data substitution $\xi$ such that $\sigma(x_i) = p_i$ for all $0 \leq i < ar(f)$, $\xi(u) = d$, $\sigma(t') = p'$ and $\xi(u') = d'$. Furthermore, for each $i \in I$, there exist a proof of $(\sigma(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma(y_i), \xi(u_i'))$ with smaller depth.

We assume acyclicity of the process-variable dependency graph (a slight variation of Definition 3.2). A process-variable dependency graph has variables as its nodes and for each $i \in I$ there exists an edge from any variable $x \in vars(t_i)$ to variable $y_i$. Hence, we can define a rank, $rank(x)$, for each variable $x$, as the maximum length of a backward chain starting from $x$ in the process-variable dependency graph. The rank of a premise is the rank of its target variable. Then, for each $x \in vars(t_i)$ of each premise $(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i')$ of the deduction rule, it holds that $rank(x) < rank(y_i)$.

We define the process substitution $\sigma'$ as follows:

$$\sigma'(x) = \begin{cases} q_i & \text{if } x = x_i \text{ for some } 0 \leq i < ar(f), \\ \sigma(x) & \text{if } x \notin X_{\mathrm{p}} \cup Y_{\mathrm{p}}. \end{cases}$$

Note that thus far this process substitution is not defined for variables from $Y_{\mathrm{p}}$. We extend the definition while proving, by induction on the rank of a premise $r$, three essential properties: for all $r$, for all $i \in I$ such that $rank(y_i) = r$,

1. $\sigma(t_i)\ \tilde{R}\ \sigma'(t_i)$;
2. $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma'(y_i), \xi(u_i'))$;
3. $\sigma(y_i)\ \tilde{R}\ \sigma'(y_i)$.

Again, we do not show the proof of the induction base ($r = 0$) as it is an instance of the proof of the induction step.

For the induction step, suppose $r \geq 1$. Let $(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i')$ for some $i \in I$ be a premise of rank $r$. First, we prove property (1). Let $x \in vars(t_i)$. We distinguish three cases:

1. $x \in X_{\mathrm{p}}$. Then $x = x_i$ for some $0 \leq i < ar(f)$. From the definition of $\sigma'$ we have that $\sigma(x) = \sigma(x_i) = p_i$ and $\sigma'(x_i) = q_i$ and we already know that $p_i\ \tilde{R}\ q_i$. Thus, we have $\sigma(x)\ \tilde{R}\ \sigma'(x)$.
2. $x \notin X_{\mathrm{p}}$ and $x \notin Y_{\mathrm{p}}$. As $\sigma(x) = \sigma'(x)$ and the identity relation is contained in $\tilde{R}$ obviously $\sigma(x)\ \tilde{R}\ \sigma'(x)$.
3. $x \in Y_{\mathrm{p}}$. Then $x = y_j$ for some $j \in I$. Obviously, also $rank(y_j) < rank(y_i)$. Thus by the induction hypothesis (property (3)) we have, $\sigma(y_j)\ \tilde{R}\ \sigma'(y_j)$. But, as $x = y_j$, we also have $\sigma(x)\ \tilde{R}\ \sigma'(x)$).

From the fact that $\sigma(x)\ \tilde{R}\ \sigma'(x)$ for all $x \in vars(t_i)$, we have, by Lemma 8.15, that $\sigma(t_i)\ \tilde{R}\ \sigma'(t_i)$; which proves property (1).

As we have a proof of smaller depth for $(\sigma(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma(y_i), \xi(u_i'))$, by the induction hypothesis, we have the existence of a process term $q_i'$ such that $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (q_i', \xi(u_i'))$ and $\sigma(y_i)\tilde{R}q_i'$. We choose $\sigma'(y_i)$ to be $q_i'$. Observe that this proves existence of an appropriate process term $\sigma'(y_i)$. Then, we also have $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma'(y_i), \xi(u_i'))$ and $\sigma(y_i)\tilde{R}\sigma'(y_i)$, which prove properties (2) and (3).

Now, we finish our reasoning using process substitution $\sigma'$ and the same data substitution and deduction rule.

Observe that indeed $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) = f(\overrightarrow{q}_{ar(f)-1}) = q$. By property (2) we have proven that there are proofs for all premises using the process substitution $\sigma'$ and data substitution $\xi$. Then, according to the same deduction rule and using $\sigma'$ instead of $\sigma$, we have $(\sigma'(f(\overrightarrow{x}_{ar(f)-1})), \xi(u)) \xrightarrow{l}_r (\sigma'(t'), \xi(u'))$. Since $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) = f(\overrightarrow{q}_{ar(f)-1}) = q$, $\xi(u) = d$ and $\xi(u') = d'$ we obtain $(q, d) \xrightarrow{l}_r (\sigma'(t'), d')$.

We only have to show that $\sigma(t')\ \tilde{R}\ \sigma'(t')$. By Lemma 8.15, it suffices to show that $\sigma(x)\ \tilde{R}\ \sigma'(x)$ for all $x \in \mathit{vars}(t')$. Three cases can be distinguished:

1. $x \in X_{\mathrm{p}}$. Then $x = x_i$ for some $0 \leq i < ar(f)$. We have that $\sigma(x_i) = p_i$ and $\sigma'(x_i) = q_i$ and we already know that $p_i\ \tilde{R}\ q_i$ and $x_i = x$. Thus, we have $\sigma(x)\ \tilde{R}\ \sigma'(x)$.
2. $x \notin X_{\mathrm{p}}$ and $x \notin Y_{\mathrm{p}}$. As $\sigma(x) = \sigma'(x)$ and the identity relation is contained in $\tilde{R}$ obviously $\sigma(x)\ \tilde{R}\ \sigma'(x)$.
3. $x \in Y_{\mathrm{p}}$. Then $x = y_j$ for some $j \in I$. We have $\sigma(y_j)\ \tilde{R}\ \sigma'(y_j)$ by property (3). But, as $x = y_j$, we also have $\sigma(x)\ \tilde{R}\ \sigma'(x)$.

So this concludes the proof of Theorem 8.13. ⊠

**Example 8.16** Consider the transition system specification of Example 8.4. Obviously, all deduction rules are in the process-tyft format, hence, by Theorem 8.13, stateless bisimilarity is a congruence for all process functions from the signature of this transition system specification.

### 8.3.2 Congruence Format for State-based Bisimilarity

In this section, we introduce a format for establishing process-congruence of state-based bisimilarity. First, we show that we cannot simply use the previously introduced process-tyft format.

**Example 8.17** Consider a transition system specification in process-tyft format, where the signature consists of three process constants $a$, $b$, and $c$, one unary process function $f$, and two data constants $d$ and $d'$ and the deduction rules are the following:

$$(1)\ \frac{}{(a, v) \xrightarrow{l} (b, d')} \qquad (2)\ \frac{}{(b, d) \xrightarrow{l} (b, d')} \qquad (3)\ \frac{}{(f(x), v) \xrightarrow{l} (x, d')}$$

Then, we have: $(a, d) \leftrightarrow_{sb} (b, d)$. On the other hand, however, it does not hold that $(f(a), d) \leftrightarrow_{sb} (f(b), d)$, since $(f(a), d)$ has an $l$-transition to $(a, d')$, while $(f(b), d)$

only has an $l$-transition to $(b, d')$ and these two states are not state-based bisimilar as the first one has an $l$-transition due to deduction rule (1), while the second one does not. Hence, state-based bisimilarity is not a process-congruence (for $f$).

In the conclusions of the deduction rules of the above example, we have transitions that (potentially) change the data state while keeping the process variable. That is the reason why we fail to have that state-based bisimilarity is a process-congruence.

We remedy this shortcoming by adding more constraints to the format. Namely, we force the links between process variables and data terms to remain consistent in each of the deduction rules as follows.

**Definition 8.18 (Sfsb)** A deduction rule *dr* of the following form

$$(dr)\ \frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') \mid i \in I\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) \xrightarrow{l}_{r} (t', u')}, s$$

is in the $\mathsf{sfsb}$ format (for $\mathsf{s}$tandard $\mathsf{f}$ormat for $\mathsf{s}$tate-$\mathsf{b}$ased bisimilarity) if it is in $\mathsf{process\text{-}tyft}$ format and satisfies the following *data-dependency* constraints:

1. If a variable $x \in X_{\mathrm{p}}$ appears in $t'$, then $u' = u$;
2. If a variable $y_i \in Y_{\mathrm{p}}$ appears in $t'$, then $u' = u_i$;
3. If a variable $x \in X_{\mathrm{p}}$ appears in some $t_i$, then $u_i = u$;
4. If a variable $y_i \in Y_{\mathrm{p}}$ appears in some $t_j$, then $u_j = u_i'$.

A transition system specification is in the $\mathsf{sfsb}$ format when all its deduction rules are.

Informally speaking, we foresee a flow of binding, as depicted below, between process variables and data terms from the source of the conclusion to the source of the premises and the target of the conclusion and from the target of the premises to the sources of other premises and finally, to the target of the conclusion.

$$\frac{\cdots \qquad (C'[x_i], u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') \qquad \cdots}{(f(x_0, \cdots, x_i, \cdots, x_{n-1}), u) \xrightarrow{l}_{r} (C[x_i], u')}$$

$$\frac{\cdots \quad (t_j, u_j) \xrightarrow{l_j}_{r_j} (y_j, u'_j) \quad \cdots \quad (C'[y_j], u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i) \quad \cdots}{(f(x_0, \cdots, x_{n-1}), u) \xrightarrow{l}_{r} (C[y_i], u')}$$

**Theorem 8.19** If a transition system specification is in the sfsb format, then state-based bisimilarity is a process-congruence for that transition system specification.

Before proving this theorem, we first define the closure of a relation under state-based congruence and give and prove a lemma that is very useful in the proof of Theorem 8.19.

**Definition 8.20** Let $R \subseteq (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}})) \times (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}}))$. We define the relation $\widehat{R} \subseteq (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}})) \times (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}}))$ to be the smallest reflexive process-congruence that contains $R$. Formally, $\widehat{R}$ is defined to be the smallest relation that satisfies:

1. $\widehat{R}$ is reflexive;
2. $R \subseteq \widehat{R}$;
3. $(f(\overrightarrow{p}_{ar(f)-1}), d)\ \widehat{R}\ (f(\overrightarrow{q}_{ar(f)-1}), d)$ for all $(f, ar(f)) \in \Sigma_{\mathrm{p}}$, $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$, and all $\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1} \in \mathcal{C}(\Sigma_{\mathrm{p}})$ such that $(p_i, d)\ \widehat{R}\ (q_i, d)$ for all $0 \leq i < ar(f)$.

**Lemma 8.21** Let $R \subseteq (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}})) \times (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}}))$, $t \in \mathcal{T}(\Sigma_{\mathrm{p}}, V_{\mathrm{p}})$, and $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$. For any process substitutions $\sigma$ and $\sigma'$ such that $(\sigma(x), d)\ \widehat{R}\ (\sigma'(x), d)$ for all $x \in vars(t)$, we have $(\sigma(t), d)\ \widehat{R}\ (\sigma'(t), d)$.

*Proof.* By induction on the structure of process term $t$. The proof is similar to the proof of Lemma 8.15 which can be consulted in [64]. ⊠

*Proof of Theorem 8.19* It suffices to prove that state-based bisimilarity is a process-congruence for each of the process functions of $\Sigma_{\mathrm{p}}$. Let $(f, ar(f)) \in \Sigma_{\mathrm{p}}$, $\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1} \in \mathcal{C}(\Sigma_{\mathrm{p}})$ and $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$. Suppose that $(p_i, d) \leftrightarrows_{sb} (q_i, d)$

for $0 \leq i < ar(f)$. This means that there are state-based bisimulation relations $R_i$ (for $0 \leq i < ar(f)$) that witness these state-based bisimilarities. Let $R$ be the union of these relations $R_i$: $R = \bigcup_{i=0}^{ar(f)-1} R_i$. Obviously $R$ is also a state-based bisimulation relation. We prove that the relation $\widehat{R}$ contains the pair $((f(\overrightarrow{p}_{ar(f)-1}), d), (f(\overrightarrow{q}_{ar(f)-1}), d))$ and that it is a state-based bisimulation relation. The first part is obvious from the definition of $\widehat{R}$.

So, we only have to prove the following for any $(p, d)\ \widehat{R}\ (q, d)$: if for an arbitrary $r$, $l$, $p'$ and $d'$, $(p, d) \xrightarrow{l}_r (p', d')$, then there exists a $q'$ such that $(q, d) \xrightarrow{l}_r (q', d')$ and $(p', d')\ \widehat{R}\ (q', d')$and vice versa for transitions of $q$. Due to symmetry, it suffices to provide the proofs for the transitions of $p$ only.

We prove this by induction on the depth of the proof of a transition. We do not show the proof for the induction base as it is an instance of the proof of the induction step where there are no premises.

For the induction step, we distinguish three cases based on the structure of the definition of $\widehat{R}$. In case the pair $((p, d), (q, d))$ is contained in the identity relation or the relation $R$, the proof is obvious (and requires no induction at all). For the remaining case, we find $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(\overrightarrow{q}_{ar(f)-1})$ for some $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1}$ such that $(p_i, d)\ \widehat{R}\ (q_i, d)$ for all $0 \leq i < ar(f)$. The last step of the proof of the transition of $p$ is due to the application of a deduction rule of the following form:

$$\frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') \mid i \in I\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) \xrightarrow{l}_r (t', u')}.$$

This means that there are a process substitution $\sigma$ and a data substitution $\xi$ such that $\sigma(x_i) = p_i$ for all $0 \leq i < n$, $\xi(u) = d$, $\sigma(t') = p'$ and $\xi(u') = d'$. Furthermore, for each $i \in I$, there exists a proof of $(\sigma(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma(y_i), \xi(u_i'))$ with smaller depth.

Since we have assumed acyclicity of the process-variable dependency graph, we can define a rank, $rank(x)$, for each variable $x$, as the maximum length of a backward chain starting from $x$ in the process-variable dependency graph. The rank of a premise is the rank of its target variable. Then, for each $x \in vars(t_i)$ of each premise $(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i')$ of the deduction rule, it holds that $rank(x) < rank(y_i)$.

We define the process substitution $\sigma'$ as follows:

$$\sigma'(x) = \begin{cases} q_i & \text{if } x = x_i \text{ for some } 0 \leq i < ar(f), \\ \sigma(x) & \text{if } x \notin X_\mathrm{p} \cup Y_\mathrm{p}. \end{cases}$$

Note that thus far this process substitution is not defined for variables from $Y_\mathrm{p}$. We extend the definition while proving, by induction on the rank of a premise $r$, three essential properties: for all $r$, for all $i \in I$ such that $rank(y_i) = r$,

1. $(\sigma(t_i), \xi(u_i)) \ \widehat{R} \ (\sigma'(t_i), \xi(u_i))$;
2. $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma'(y_i), \xi(u_i'))$;
3. $(\sigma(y_i), \xi(u_i')) \ \widehat{R} \ (\sigma'(y_i), \xi(u_i'))$.

Again, we do not show the proof of the induction base ($r = 0$) as it is an instance of the proof of the induction step.

For the induction step, suppose $r \geq 1$. Let $(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i')$ for some $i \in I$ be a premise of rank $r$. First, we prove property (1). Let $x \in \mathit{vars}(t_i)$. We show that $(\sigma(x_i), \xi(u_i)) \ \widehat{R} \ (\sigma'(x_i), \xi(u_i)))$. To do this, the following three cases are distinguished:

1. $x \in X_{\mathrm{p}}$. Then $x = x_i$ for some $0 \leq i < ar(f)$. The source of the premise has process variable $x_i \in X_{\mathrm{p}}$; hence, by data-dependency constraint 3, $u_i = u$. We also have that $\sigma(x_i) = p_i$ and $\sigma'(x_i) = q_i$ and we already know that $(p_i, d) \ \widehat{R} \ (q_i, d)$ and $\xi(u) = d$. Thus, we have $(\sigma(x_i), \xi(u_i)) \ \widehat{R} \ (\sigma'(x_i), \xi(u_i))$, i.e., $(\sigma(x), \xi(u_i)) \ \widehat{R} \ (\sigma'(x), \xi(u_i))$.
2. $x \notin X_{\mathrm{p}}$ and $x \notin Y_{\mathrm{p}}$. As $\sigma(x) = \sigma'(x)$ and the identity relation is contained in $\widehat{R}$ obviously $(\sigma(x), \xi(u')) \ \widehat{R} \ (\sigma'(x), \xi(u'))$.
3. $x \in Y_{\mathrm{p}}$. Then $x = y_j$ for some $j \in I$. Hence, by data-dependency constraint 4, $u_i = u_j'$. Obviously, also $\mathit{rank}(y_j) < \mathit{rank}(y_i)$. Thus by the induction hypothesis (property (2)) we have, $(\sigma(y_j), \xi(u_j')) \ \widehat{R} \ (\sigma'(y_j), \xi(u_j'))$. But, as $x = y_j$, and $u_j' = u_i$, we also have $(\sigma(x), \xi(u_i)) \ \widehat{R} \ (\sigma'(x), \xi(u_i))$.

From the fact that $(\sigma(x), \xi(u_i)) \ \widehat{R} \ (\sigma'(x), \xi(u_i)))$ for all $x \in \mathit{vars}(t_i)$, by Lemma 8.21, we have $(\sigma(t_i), \xi(u_i)) \ \widehat{R} \ (\sigma'(t_i), \xi(u_i))$; which proves property (1).

As we have a proof of smaller depth for $(\sigma(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma(y_i), \xi(u_i'))$, by the induction hypothesis, we have the existence of a process term $q_i'$ such that $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (q_i', \xi(u_i'))$ and $(\sigma(y_i), \xi(u_i')) \ \widehat{R} \ (q_i', \xi(u_i'))$. We choose $\sigma'(y_i)$ to be $q_i'$. Observe that this proves existence of an appropriate process term $\sigma'(y_i)$. Then, obviously, we also have $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma'(y_i), \xi(u_i'))$ and $(\sigma(y_i), \xi(u_i')) \ \widehat{R} \ (\sigma'(y_i), \xi(u_i'))$, which prove properties (2) and (3).

Now, we finish our reasoning using process substitution $\sigma'$ and the same data substitution and deduction rule. Observe that indeed $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) = f(\overrightarrow{q}_{ar(f)-1}) = q$. By property (2) we have proven that there are proofs for all premises using the process substitution $\sigma'$ and data substitution $\xi$. Then, according to the same deduction rule and using $\sigma'$ instead of $\sigma$, we have $(\sigma'(f(x_0, \ldots, x_{n-1})), \xi(u)) \xrightarrow{l}_{r}$

$(\sigma'(t'), \xi(u'))$. Since $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) = f(\overrightarrow{q}_{ar(f)-1}) = q$, $\xi(u) = d$ and $\xi(u') = d'$ we obtain $(q, d) \xrightarrow{l}_r (\sigma'(t'), d')$.

We only have to show that $(\sigma(t'), d')\ \widehat{R}\ (\sigma'(t'), d')$. By Lemma 8.21, it suffices to show that $(\sigma(x), d')\ \widehat{R}\ (\sigma'(x), d')$ for all $x \in \mathit{vars}(t')$. Three cases can be distinguished:

1. $x \in X_{\mathrm{p}}$. Then $x = x_i$ for some $0 \leq i < n$. Hence, $x \in X_{\mathrm{p}}$ and by data-dependency constraint 1, $u = u'$. Hence, $d = \xi(u) = \xi(u') = d'$. We also have that $\sigma(x_i) = p_i$ and $\sigma'(x_i) = q_i$ and we already know that $(p_i, d)\ \widehat{R}\ (q_i, d)$, $x_i = x$, and $d = d'$. Thus, we have $(\sigma(x), d')\ \widehat{R}\ (\sigma'(x), d')$.
2. $x \notin X_{\mathrm{p}}$ and $x \notin Y_{\mathrm{p}}$. As $\sigma(x) = \sigma'(x)$ and the identity relation is contained in $\widehat{R}$ obviously $(\sigma(x), d')\ \widehat{R}\ (\sigma'(x), d')$.
3. $x \in Y_{\mathrm{p}}$. Then $x = y_j$ for some $j \in I$. Hence, by data-dependency constraint 2, $u'_j = u'$. We obtain $d' = \xi(u') = \xi(u'_j)$. By property (3) we have, $(\sigma(y_j), \xi(u'_j))\ \widehat{R}\ (\sigma'(y_j), \xi(u'_j))$. But, as $x = y_j$, and $\xi(u'_j) = d'$, we also have $(\sigma(x), d')\ \widehat{R}\ (\sigma'(x), d')$.

So this concludes the proof of Theorem 8.19. ⊠

Next, we show that if the proposed format is relaxed by dropping each of the syntactic constraints, the congruence result is lost. The first example shows that we cannot remove data-dependency constraint 1.

**Example 8.22** Consider a transition system specification, where the process signature consists of process constants $a$ and $b$, and a unary function symbol $f$; the data signature consists of data constants $d$ and $d'$; and the following deduction rules:

$$(1)\ \frac{}{(a, d') \xrightarrow{l} (a, d')} \qquad (2)\ \frac{}{(f(x), d) \xrightarrow{l} (x, d')}$$

These deduction rules are in the process-tyft format. Data-dependency constraint 1 is not satisfied by deduction rule (2) as in the target of the conclusion the variable $x \in X_{\mathrm{p}}$ appears but $d \neq d'$. The other data-dependency constraints are indeed satisfied.

The process-congruence result fails on the above specification. As both $(a, d)$ and $(b, d)$ cannot perform any transitions, we have $(a, d) \mathrel{\underline{\leftrightarrow}}_{sb} (b, d)$. However, it does not hold that $(f(a), d) \mathrel{\underline{\leftrightarrow}}_{sb} (f(b), d)$ since the former state can perform a transition due to deduction rule (2) to $(a, d')$, while the latter is forced to make the same transition to $(b, d')$ and it clearly does not hold that $(a, d') \mathrel{\underline{\leftrightarrow}}_{sb} (b, d')$ (see deduction rule (1)).

The next example shows that we cannot remove data-dependency constraint 2.

**Example 8.23** Consider a process signature consisting of process constants $a$ and $b$ and a unary process function $f$; a data signature consisting of data constants $d$ and $d'$; and a transition system specification with the following deduction rules:

$$(1)\ \frac{}{(a,v) \xrightarrow{l} (a,v)} \qquad (2)\ \frac{}{(b,d) \xrightarrow{l} (b,d)} \qquad (3)\ \frac{(x,d) \xrightarrow{l} (y,d)}{(f(x),d) \xrightarrow{l} (y,d')}$$

These deduction rules are in process-tyft format and all data-dependency constraints, except for constraint 2, which is violated by deduction rule (3). This violation results in breaking the process-congruence result. Two states $(a,d)$ and $(b,d)$ are state-based bisimilar. However, $(f(a),d)$ is not state-based bisimilar to $(f(b),d)$ since the former can perform a transition using deduction rule (3) to $(a,d')$, while the latter performs a similar transition to $(b,d')$. These two states are not state-based bisimilar as the former performs an $l$-transition and the latter deadlocks.

The next example shows that we cannot remove data-dependency constraint 3.

**Example 8.24** Consider a transition system specification, where the process signature consists of process constants $a$ and $b$, and a unary function symbol $f$; the data signature consists of data constants $d$ and $d'$; and the following deduction rules:

$$(1)\ \frac{}{(a,d') \xrightarrow{l} (a,d')} \qquad (2)\ \frac{(x,d') \xrightarrow{l} (y,v')}{(f(x),v) \xrightarrow{l} (x,v)}$$

The deduction rules are in the process-tyft format and satisfy data-dependency constraints 1, 2, and 4. Data-dependency constraint 3 is violated in deduction rule (2) since variable $x \in X_\mathrm{p}$ appears in the source of the premise but $d' \neq v$.

For this transition system specification, state-based bisimilarity is not a process-congruence. Although we have $(a,d) \leftrightarrow_{sb} (b,d)$ (because both states cannot make a transition), it is not the case that $(f(a),d)$ and $(f(b),d)$ are state-based bisimilar, since the former state can make a transition due to deduction rule (2) while the latter cannot make any transition.

The next example shows that we cannot remove data-dependency constraint 4.

**Example 8.25** Consider a process signature consisting of process constants $a$ and $b$ and a unary process function $f$; a data signature consisting of data constants $d$

and $d'$; and a transition system specification with the following deduction rules:

$$(1)\ \frac{}{(a,v)\xrightarrow{l}(a,v)} \qquad (2)\ \frac{}{(b,d)\xrightarrow{l}(b,d)}$$

$$(3)\ \frac{(x,d)\xrightarrow{l}(y,d) \quad (y,d')\xrightarrow{l}(y',d')}{(f(x),d)\xrightarrow{l}(y',d')}$$

The above deduction rules are in the process-tyft format and satisfy all data-dependency constraints apart from constraint 4. Deduction rule (3) breaks this constraint in the source of its second premise. This also turns out to be harmful for the congruence property, since we have $(a,d) \mathrel{\underline{\leftrightarrow}_{sb}} (b,d)$ but not $(f(a),d) \mathrel{\underline{\leftrightarrow}_{sb}} (f(b),d)$ because deduction rule (3) allows for a transition of the former but not the latter.

### 8.3.3 Congruence Format for Initially Stateless Bisimilarity

Later, when comparing congruence conditions for the different notions of bisimilarity, we show that the sfsb format works perfectly well for initially stateless bisimilarity. However, it may turn out to be too restrictive in applications. The following example shows a common problem in this regard.

**Example 8.26** Consider the following transition system specification (with process constants $a$ and $b$, unary process function $f$, and data constants $d$ and $d'$) and the following deduction rules:

$$(1)\ \frac{}{(a,v)\xrightarrow{l}(a,v)} \qquad (2)\ \frac{}{(b,d)\xrightarrow{l}(b,d)} \qquad (3)\ \frac{(x_0,v)\xrightarrow{l}(y,v)}{(f(x_0,x_1),v)\xrightarrow{l}(x_1,d')}$$

This transition system specification does not satisfy the sfsb format and state-based bisimilarity is not a congruence (since $(a,d) \mathrel{\underline{\leftrightarrow}_{sb}} (b,d)$, but it does not hold that $(f(b,a),d) \mathrel{\underline{\leftrightarrow}_{sb}} (f(b,b),d)$). However, it can be checked that initially stateless bisimilarity is indeed a congruence. The reason is that although deduction rule (3) violates data-dependency constraint 1, the violating change in the data is harmless since $x_1$'s are now related using all data states including $d'$ (e.g., the above counterexample does not work anymore since it does not hold that $a \mathrel{\underline{\leftrightarrow}_{isl}} b$).

This gives us some clue that, for initially stateless bisimilarity, we may weaken the data-dependency constraints.

**Definition 8.27 (Sfisl)** A deduction rule $dr$ of the following form

$$(dr)\ \frac{\{(t_i,u_i)\xrightarrow{l_i}_{r_i}(y_i,u_i') \mid i\in I\}}{(f(\overrightarrow{x}_{ar(f)-1}),u)\xrightarrow{l}_{r}(t',u')},$$

is in the sfisl format (for standard format for initially stateless bisimilarity) if it is in the process-tyft format and satisfies the following local (relaxed) data-dependency constraints:

1. If a variable $y_i \in Y_{\mathrm{p}}$ appears in $t'$, then $u' = u_i$;
2. If a variable $y_i \in Y_{\mathrm{p}}$ appears in some $t_j$, then $u_j = u'_i$.

Note that the above two constraints are the same as constraints 2 and 4 in Definition 8.18. However, the two other data-dependency constraints of Definition 8.18 that were required for variables from the set $X_{\mathrm{p}}$, need not be satisfied for this format anymore. The reason of violating these constraints is that we rely on the fact that certain positions are instantiated by process terms that are related for all possible data. To formalize this concept, first we define positions for which the two constraints are violated and then we check the global consequences of this violation.

**Definition 8.28** For a deduction rule $(dr)$ of the above form, variable $x \in X_{\mathrm{p}}$ is called *unresolved* if

$$\exists_{i \in I}\ (x \in vars(t_i) \Rightarrow u \neq u_i) \vee (x \in vars(t') \Rightarrow u \neq u').$$

We define $X_{\mathrm{p}}^{\mathrm{u}}$ to be the set of unresolved variables from the set $X_{\mathrm{p}}$.

For each process function $f$, we define a set $IV_f$ that contains indices of $f$ for which we need initially stateless bisimilarity because a data-dependency is violated with respect to the variable that occurs in that position in the source of the conclusion. The set $IV_f$ contains at least the indices of the unresolved variables of the $f$-defining deduction rules, but it may contain more indices due to the use of $f$ in other deduction rules in the target of the conclusion or the source of a premise.

**Definition 8.29** For a given transition system specification in the process-tyft format, we define, for all $(f, ar(f)) \in \Sigma_{\mathrm{p}}$, $IV_f$ as a minimal set that satisfies, for all $f$-defining deduction rules $dr$:

1. the indices of unresolved variables (i.e., variables from $X_{\mathrm{p}}^{\mathrm{u}}$) of $dr$ are in $IV_f$;
2. for all $n$-ary process functions $g \in \Sigma_{\mathrm{p}}$: for each occurrence of a process term $g(t_0, \ldots, t_{n-1})$ in the source of a premise or the target of the conclusion of $dr$:
$$\forall_{i \in IV_g}\ \forall_{x \in vars(t_i)}\ \exists_{j \in IV_f}\ x = x_j.$$

Note that with the above definition, it is possible that such a set does not exist. In such cases, the two global data-dependency constraints given above cannot be consistently established. The following two examples illustrate existence and absence of $IV_f$.

**Example 8.30** Consider the transition system specification of Example 8.26, in deduction rule (3), variable $x_1$ is unresolved, and thus $1 \in IV_f$. For the deduction rules defining the two constants $a$ and $b$, there are no unresolved variables and $IV_a = IV_b = \emptyset$. The second global condition is trivially satisfied for all $IV$ sets.

**Example 8.31** Consider the following transition system specification with process constants $a$, $b$ and $c$, unary process functions $f$ and $g$ and data constants $d$, $d'$ and $d''$.

$$(1)\ \frac{}{(a,d) \xrightarrow{l} (b,d')} \qquad (2)\ \frac{}{(b,d) \xrightarrow{l} (c,d')} \qquad (3)\ \frac{}{(c,d'') \xrightarrow{l} (c,d'')}$$

$$(4)\ \frac{(x_0,d) \xrightarrow{l} (y_0,d')}{(f(x_0),d) \xrightarrow{l} (g(y_0),d')} \qquad (5)\ \frac{}{(g(x_0),d') \xrightarrow{l} (x_0,d'')}$$

In deduction rule (5), variable $x_0$ is unresolved and hence $0 \in IV_g$. However, global constraint 2 requires that in deduction rule (4), $y_0 = x_0$ which is a contradiction (since $0 \in IV_g$, $y_0 \in vars(y_0)$ and $f$ is a unary process function). Hence, we may conclude that no consistent $IV_g$ exists. In fact, one can check that initially stateless is not a congruence for the above transition system specification, as it holds that $a \leftrightarroweq_{isl} b$ but not $f(a) \leftrightarroweq_{isl} f(b)$ ($(f(a),d)$ after two steps arrives in $(b,d'')$ which deadlocks but $(f(b),d)$ arrives in $(c,d'')$ which can perform self-transitions).

**Definition 8.32 (Sfisl)** A transition system specification is in the sfisl format when all its deduction rules are in the sfisl format and furthermore for each process function $f$ the set $IV_f$ exists.

Informally, this means that a deduction rule may change the data state associated with a process term (arbitrarily) if according to the other rules, the process term is guaranteed to be among the initial arguments of the topmost process function (thus, benefitting from the initially stateless bisimilarity assumption). The positions of a process function $f$ benefitting from the initially stateless bisimilarity assumption are thus denoted by $IV_f$.

**Theorem 8.33** If a transition system specification is in the sfisl format, then initially stateless bisimilarity is a congruence for that transition system specification.

Before we prove this theorem, we first define the closure of a relation under initially stateless congruence and give and prove a lemma that is very useful in the proof of Theorem 8.33.

**Definition 8.34 (Closure With Initially Stateless Congruence)** Let $R \subseteq (\mathcal{C}(\Sigma_\mathrm{p}) \times \mathcal{C}(\Sigma_\mathrm{d})) \times (\mathcal{C}(\Sigma_\mathrm{p}) \times \mathcal{C}(\Sigma_\mathrm{d}))$. We define the relation $\overline{R} \subseteq (\mathcal{C}(\Sigma_\mathrm{p}) \times \mathcal{C}(\Sigma_\mathrm{d})) \times (\mathcal{C}(\Sigma_\mathrm{p}) \times \mathcal{C}(\Sigma_\mathrm{d}))$ to be the smallest relation that satisfies:

1. $\overline{R}$ is reflexive;
2. $R \subseteq \overline{R}$;
3. $(f(\overrightarrow{p}_{ar(f)-1}), d)\ \overline{R}\ (f(\overrightarrow{q}_{ar(f)-1}), d)$ for all $(f, ar(f)) \in \Sigma_{\mathrm{p}}$, $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$, and all $\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1} \in \mathcal{C}(\Sigma_{\mathrm{p}})$ such that
   (a) $\forall_{i \notin IV_f}\ (p_i, d)\ \overline{R}\ (q_i, d)$;
   (b) $\forall_{i \in IV_f, d' \in \mathcal{C}(\Sigma_{\mathrm{d}})}\ (p_i, d')\ \overline{R}\ (q_i, d')$.

For a process term $t$, we define the set $V(t)$ to be the set of variables that appear in the places indicated by the sets $IV_f$ (for all $f$).

$$\begin{array}{lcl} V(x) & = & \emptyset, \\ V(f(\overrightarrow{t}_{ar(f)-1})) & = & \displaystyle\bigcup_{0 \leq i < ar(f), i \in IV_f} vars(t_i) \cup \bigcup_{0 \leq i < ar(f), i \notin IV_f} V(t_i). \end{array}$$

**Lemma 8.35** Let $R \subseteq (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}})) \times (\mathcal{C}(\Sigma_{\mathrm{p}}) \times \mathcal{C}(\Sigma_{\mathrm{d}}))$, $t \in \mathcal{T}(\Sigma_{\mathrm{p}}, V_{\mathrm{p}})$, $d \in \mathcal{C}(\Sigma_{\mathrm{d}})$. For any two process substitutions $\sigma$ and $\sigma'$ such that

1. $((\sigma(x), d'), (\sigma'(x), d')) \in \overline{R}$ for all $x \in V(t)$, $d' \in \mathcal{C}(\Sigma_{\mathrm{d}})$, and
2. $((\sigma(x), d), (\sigma'(x), d)) \in \overline{R}$ for all $x \in vars(t) \setminus V(t)$;

we have $(\sigma(t), d)\ \overline{R}\ (\sigma'(t), d)$.

*Proof.* By induction on the structure of process term $t$. In case $t$ is a variable, say $x$, we obtain $\sigma(t) = \sigma(x)$ and $\sigma'(t) = \sigma'(x)$ and $V(t) = V(x) = \emptyset$. As $x \in vars(t) \setminus V(t)$, we have $((\sigma(x), d), (\sigma'(x), d)) \in \overline{R}$ and therefore $(\sigma(t), d)\ \overline{R}\ (\sigma'(t), d)$ as well.

In case $t$ is a constant, say $c$, we obtain $\sigma(t) = \sigma(c) = c = \sigma'(c) = \sigma'(t)$. Then, from reflexivity of $\overline{R}$, it follows immediately that $(\sigma(t), d)\ \overline{R}\ (\sigma'(t), d)$.

Finally, consider the case where $t = f(\overrightarrow{t}_{ar(f)-1})$ for some $(f, ar(f)) \in \Sigma_{\mathrm{p}}$ and $\overrightarrow{t}_{ar(f)-1} \in \mathcal{T}(\Sigma_{\mathrm{p}}, V_{\mathrm{p}})$. If we prove

$$((\sigma(t_i), d), (\sigma'(t_i), d)) \in \overline{R} \tag{8.1}$$

for all $i \notin IV_f$, and

$$((\sigma(t_i), d'), (\sigma'(t_i), d')) \in \overline{R} \tag{8.2}$$

for all $i \in IV_f$ and $d' \in \mathcal{C}(\Sigma_{\mathrm{d}})$, then $((\sigma(t), d), (\sigma'(t), d)) \in \overline{R}$ according to Definition 8.34.

For the first part, assume that $i \notin IV_f$. Then, by definition of $V$, we have $V(t_i) \subseteq V(t)$. Therefore, by the first assumption on $\sigma$ and $\sigma'$ of Lemma 8.35, we have $(\sigma(x), d')\ \overline{R}\ (\sigma'(x), d')$ for all $x \in V(t_i)$ and $d' \in \mathcal{C}(\Sigma_\mathrm{d})$. By the first and second assumption and the fact that $vars(t_i) \setminus V(t_i) \subseteq vars(t)$, we have $(\sigma(x), d)\ \overline{R}\ (\sigma'(x), d)$ for all $x \in vars(t_i) \setminus V(t_i)$. Thus, by the induction hypothesis, we have $(\sigma(t_i), d)\ \overline{R}\ (\sigma'(t_i), d)$.

For the second part, assume that $i \in IV_f$ and that $d' \in \mathcal{C}(\Sigma_\mathrm{d})$. From the definition of $V$ we obtain $vars(t_i) \subseteq V(t)$. Hence, by the first assumption on $\sigma$ and $\sigma'$ of Lemma 8.35, we have $((\sigma(x), d'), (\sigma'(x), d')) \in \overline{R}$ for all $x \in vars(t_i)$. Thus, by the induction hypothesis, we have $((\sigma(t_i), d'), (\sigma'(t_i), d')) \in \overline{R}$. ⊠

*Proof of Theorem 8.33.* It suffices to prove that initially stateless bisimilarity is a congruence for each of the process functions of $\Sigma_\mathrm{p}$. Let $(f, ar(f)) \in \Sigma_\mathrm{p}$, $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1} \in \mathcal{C}(\Sigma_\mathrm{p})$ and $d \in \mathcal{C}(\Sigma_\mathrm{d})$. Suppose that $\overrightarrow{p}_{ar(f)-1} \underline{\leftrightarrow}_{isl} \overrightarrow{p}_{ar(q)-1}$. This means that there are state-based bisimulation relations $R_i$ (for $0 \leq i < ar(f)$) such that $(p_i, d)\ R_i\ (q_i, d)$ for all $d \in \mathcal{C}(\Sigma_\mathrm{d})$. Let $R$ be the union of these relations $R_i$: $R = \bigcup_{i=0}^{ar(f)-1} R_i$. Obviously $R$ is also a state-based bisimulation relation. We prove that the relation $\overline{R}$ contains the pair $((f(\overrightarrow{p}_{ar(f)-1}), d), (f(\overrightarrow{q}_{ar(f)-1}), d))$, for all $d \in \mathcal{C}(\Sigma_\mathrm{d})$, and that it is a state-based bisimulation relation.

As $(p_i, d)\ R_i\ (q_i, d))$ and $R_i \subseteq R \subseteq \overline{R}$, for all $0 \leq i < ar(f)$ and all $d \in \mathcal{C}(\Sigma_\mathrm{d})$, it follows that $(p_i, d)\ \overline{R}\ (q_i, d)$, for all $0 \leq i < ar(f)$ and all $d \in \mathcal{C}(\Sigma_\mathrm{d})$. Hence, by the definition of $\overline{R}$ obviously also $(f(\overrightarrow{p}_{ar(f)-1}), d)\ \overline{R}\ (f(\overrightarrow{q}_{ar(f)-1}), d))$, for $d \in \mathcal{C}(\Sigma_\mathrm{d})$.

So, we only have to prove the following for any $((p, d), (q, d)) \in \overline{R}$: if for arbitrary $r$, $l$, $p'$ and $d'$, $(p, d) \xrightarrow{l}_r (p', d')$, then there exists a $q'$ such that $(q, d) \xrightarrow{l}_r (q', d')$ and $((p', d'), (q', d')) \in \overline{R}$ and vice versa for transitions of $q$. Due to symmetry, it suffices to provide the proofs for the transitions of $p$ only.

We prove this by induction on the depth of the proof of a transition. We do not show the proof for the induction base as it is an instance of the proof of the induction step where there are no premises.

For the induction step, we distinguish three cases based on the structure of the definition of $\overline{R}$. In case the pair $((p, d), (q, d))$ is contained in $\overline{R}$ due to reflexivity of $\overline{R}$ or due to the requirement that $\overline{R}$ contains $R$, the proof is obvious (and requires no induction at all). For the remaining case, we find $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(\overrightarrow{q}_{ar(f)-1})$ for some $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1}$ such that

$$\forall_{i \notin IV_f}\ ((p_i, d), (q_i, d)) \in \overline{R}, \tag{8.3}$$

and

$$\forall_{i \in IV_f, d' \in \mathcal{C}(\Sigma_\mathrm{d})}\ ((p_i, d'), (q_i, d')) \in \overline{R}. \tag{8.4}$$

The last step of the proof of the transition of $p$ is due to the application of a deduction rule of the following form:

$$\frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u'_i) \mid i \in I\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) \xrightarrow{l}_r (t', u')}.$$

This means that there are a process substitution $\sigma$ and a data substitution $\xi$ such that $\sigma(x_i) = p_i$ for all $0 \leq i < ar(f)$, $\xi(u) = d$, $\sigma(t') = p'$ and $\xi(u') = d'$. Furthermore, for each $i \in I$, there exist a proof of $(\sigma(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma(y_i), \xi(u'_i))$ with smaller depth.

Since we have assumed acyclicity of the process-variable dependency graph, we can define a rank, $rank(x)$, for each variable $x$, as the maximum length of a backward chain starting from $x$ in the process-variable dependency graph. The rank of a premise is the rank of its target variable. Then, for each $x \in vars(t_i)$ of each premise $(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u'_i)$ of the deduction rule, it holds that $rank(x) < rank(y_i)$.

We define the process substitution $\sigma'$ as follows:

$$\sigma'(x) = \begin{cases} q_i & \text{if } x = x_i \text{ for some } 0 \leq i < ar(f), \\ \sigma(x) & \text{if } x \notin X_{\mathrm{p}} \cup Y_{\mathrm{p}}. \end{cases}$$

Note that thus far this process substitution is not defined for variables from $Y_{\mathrm{p}}$. We extend this definition while proving, by induction on the rank of a premise $r$, three essential properties: for all $r$, for all $i \in I$ such that $rank(y_i) = r$,

1. $((\sigma(t_i), \xi(u_i)), (\sigma'(t_i), \xi(u_i))) \in \overline{R}$;
2. $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma'(y_i), \xi(u'_i))$;
3. $((\sigma(y_i), \xi(u'_i)), (\sigma'(y_i), \xi(u'_i))) \in \overline{R}$.

Again, we do not show the proof of the induction base ($r = 0$) as it is an instance of the proof of the induction step.

For the induction step, suppose $r \geq 1$. Let $(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u'_i)$ for some $i \in I$ be a premise of rank $r$. First, we prove property (1). We aim at using Lemma 8.35. Hence we prove

$$\forall_{x \in vars(t_i) \setminus V(t_i)} \ ((\sigma(x), \xi(u_i)), (\sigma'(x), \xi(u_i))) \in \overline{R} \tag{8.5}$$

and

$$\forall_{x \in V(t_i), d'' \in \mathcal{C}(\Sigma_{\mathrm{d}})} \ ((\sigma(x), d''), (\sigma'(x), d'')) \in \overline{R} \tag{8.6}$$

by induction on the structure of term $t_i$.

1. Suppose that $t_i$ is a variable, say $x$. Then $vars(t_i) \setminus V(t_i) = \{x\} \setminus \emptyset = \{x\}$. For the first property, we distinguish three cases:
   - $x \notin X_{\mathrm{p}}$ and $x \notin Y_{\mathrm{p}}$. Then, we have $\sigma(t_i) = \sigma'(t_i)$. Since $\overline{R}$ is reflexive we obtain $((\sigma(x), \xi(u_i)), (\sigma'(x), \xi(u_i))) \in \overline{R}$.
   - $x \in Y_{\mathrm{p}}$. Then $x = y_j$ for some $j \in I$. Hence, by local data-dependency constraint 2, $u_i = u'_j$. Observe that $rank(y_j) < r$. By the induction hypothesis (property (3)), we have $(\sigma(y_j), \xi(u'_j))\ \overline{R}\ (\sigma'(y_j), \xi(u'_j))$. Hence, as $y_j = x$ and $\xi(u_j) = \xi(u_i)$, we have $(\sigma(x), \xi(u_i))\ \overline{R}\ (\sigma'(x), \xi(u_i))$.
   - $x \in X_{\mathrm{p}}$. Then, $x = x_j$ for some $0 \leq j < n$. We distinguish two cases:
     - $j \in IV_f$. Then, we use assumption (8.4) to obtain the desired $(\sigma(x), \xi(u_j))\ \overline{R}\ (\sigma'(x), \xi(u_j))$;
     - $j \notin IV_f$. Then by assumption (8.3) we have $(p_j, d)\ \overline{R}\ (q_j, d)$. By definition of $IV$ we obtain that $x_j$ is not an unresolved variable. Hence, by definition of unresolved variables, we have $u_i = u$. Hence $d = \xi(u) = \xi(u_i)$. Thus, we have $(\sigma(x), \xi(u_i))\ \overline{R}\ (\sigma'(x), \xi(u_i)))$.

   The second property holds trivially, as $V(t_i) = \emptyset$.
2. Suppose that $t_i$ is a process constant, say $c$. Then both properties hold trivially, as $vars(t_i) = \emptyset$ and $V(t_i) = \emptyset$.
3. Suppose that $t_i = g(\overrightarrow{t'}_{ar(g)-1})$ for some $(g, ar(g)) \in \Sigma_{\mathrm{p}}$ and $\overrightarrow{t'}_{ar(g)-1} \in \mathcal{T}(\Sigma_{\mathrm{p}}, V_{\mathrm{p}})$. For the first property observe that $x \in vars(t_i) \setminus V(t_i)$ implies that $x \in vars(t'_j) \setminus V(t'_j)$ for some $j \notin IV_g$. By the induction hypothesis (first property), we then have $(\sigma(x), \xi(u_i))\ \overline{R}\ (\sigma'(x), \xi(u_i))$.

   For the second property observe that $x \in V(t_i)$ implies (1) $x \in vars(t'_j)$ for some $0 \leq j < ar(g)$ such that $j \in IV_g$; or (2) $x \in V(t'_j)$ for some $j \notin IV_g$. In the first case, the global data-dependency constraint requires that $x = x_k$ for some $0 \leq k < n$ such that $k \in IV_f$. We have $\sigma(x) = p_k$ and $\sigma'(x) = q_k$. Using assumption (8.4) we then obtain $(\sigma(x), d'')\ \overline{R}\ (\sigma'(x), d'')$ for all $d'' \in \mathcal{C}(\Sigma_{\mathrm{d}})$. In the second case, by the induction hypothesis (second property), we have $(\sigma(x), d'')\ \overline{R}\ (\sigma'(x), d'')$ for all $d'' \in \mathcal{C}(\Sigma_{\mathrm{d}})$.

From property (1), we have that $(\sigma(t_i), \xi(u_i))\ \overline{R}\ (\sigma'(t_i), \xi(u_i))$. We also have a proof of smaller depth for $(\sigma(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (\sigma(y_i), \xi(u'_i))$. Then, by the induction hypothesis, we have the existence of a process term $q'_i$ such that $(\sigma'(t_i), \xi(u_i)) \xrightarrow{l_i}_{r_i} (q'_i, \xi(u'_i))$ and $((\sigma(y_i), \xi(u'_i)), (q'_i, \xi(u'_i))) \in \overline{R}$. We choose $\sigma'(y_i)$ to be $q'_i$. Observe that this proves existence of an appropriate process term $\sigma'(y_i)$. This concludes the proof of properties (2) and (3).

Now, we finish our reasoning using process substitution $\sigma'$ and the same data substitution and deduction rule. Observe that indeed $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) = f(\overrightarrow{q}_{ar(f)-1})$

$= q$. By property (2) we have proven that there exist proofs for all premises using the process substitution $\sigma'$ and data substitution $\xi$. Then, according to the same deduction rule and using $\sigma'$ instead of $\sigma$, we have $(\sigma'(f(\overrightarrow{x}_{ar(f)-1})), \xi(u)) \xrightarrow{l}_r (\sigma'(t'), \xi(u'))$. Since $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) = f(\overrightarrow{q}_{ar(f)-1}) = q$, $\xi(u) = d$ and $\xi(u') = d'$ we obtain $(q, d) \xrightarrow{l}_r (\sigma'(t'), d')$.

We only have to show that $(\sigma(t'), d')\overline{R}(\sigma'(t'), d')$. We aim at using Lemma 8.35. Hence we prove

$$\forall_{x \in vars(t') \setminus V(t')} \; (\sigma(x), d') \; \overline{R} \; (\sigma'(x), d') \tag{8.7}$$

and

$$\forall_{x \in V(t'), d'' \in \mathcal{C}(\Sigma_{\mathrm{d}})} \; (\sigma(x), d'') \; \overline{R} \; (\sigma'(x), d'')) \tag{8.8}$$

by induction on the structure of term $t'$.

1. Suppose that $t'$ is a variable, say $x$. Then $vars(t') \setminus V(t') = \{x\} \setminus \emptyset = \{x\}$. For the first property, we distinguish three cases:

   - $x \notin X_{\mathrm{p}}$ and $x \notin Y_{\mathrm{p}}$. Then, we have $\sigma(t_i) = \sigma'(t_i)$. Since $\overline{R}$ is reflexive we obtain $(\sigma(x), d') \; \overline{R} \; (\sigma'(x), d')$.
   - $x \in Y_{\mathrm{p}}$. Then $x = y_j$ for some $j \in I$. Hence, by local data-dependency constraint 1, $u' = u'_j$. By property (3), we have $(\sigma(y_j), \xi(u'_j)) \; \overline{R} \; (\sigma'(y_j), \xi(u'_j)))$. Hence, as $y_j = x$ and $\xi(u_j) = \xi(u') = d'$, we have $(\sigma(x), d') \; \overline{R} \; (\sigma'(x), d')$.
   - $x \in X_{\mathrm{p}}$. Then, $x = x_j$ for some $0 \leq j < n$. We distinguish two cases:
     - $j \in IV_f$. Then, we use assumption (8.4) to obtain the desired $((\sigma(x), d')), (\sigma'(x), d')) \in \overline{R}$;
     - $j \notin IV_f$, then by assumption (8.3) we have $((p_j, d), (q_j, d)) \in \overline{R}$. By definition of $IV$ we obtain that $x_j$ is not an unresolved variable. Hence, by definition of unresolved variables, we have $u' = u$. Hence $d = \xi(u) = \xi(u') = d'$. Thus, we have $(\sigma(x), d') \; \overline{R} \; (\sigma'(x), d')$.

   The second property holds trivially, as $V(t') = \emptyset$.

2. Suppose that $t'$ is a process constant, say $c$. Then both properties hold trivially, as $vars(t') = \emptyset$ and $V(t') = \emptyset$.

3. $t' = g(\overrightarrow{t'}_{ar(g)-1})$ for some process function $(g, ar(g)) \in \Sigma_{\mathrm{p}}$ and $t'_j \in T(\Sigma_{\mathrm{p}})$ for $0 \leq j < ar(g)$. For the first property observe that $x \in vars(t') \setminus V(t')$ implies that $x \in vars(t'_j) \setminus V(t'_j)$ for some $j \notin IV_g$. By the induction hypothesis (first property), we then have $(\sigma(x), d') \; \overline{R} \; (\sigma'(x), d')$.

   For the second property observe that $x \in V(t')$ implies (1) $x \in vars(t'_j)$ for some $0 \leq j < ar(g)$ such that $j \in IV_g$; or (2) $x \in V(t'_j)$ for some $j \notin IV_g$. In the first case, the global data-dependency constraint requires that $x = x_k$

for some $0 \leq k < ar(f)$ such that $k \in IV_f$. We have $\sigma(x) = p_k$ and $\sigma'(x) = q_k$. Using assumption (8.4) we then obtain $(\sigma(x), d'')\ \overline{R}\ (\sigma'(x), d'')$ for all $d'' \in \mathcal{C}(\Sigma_{\mathrm{d}})$. In the second case, by the induction hypothesis (second property), we have $(\sigma(x), d'')\ \overline{R}\ (\sigma'(x), d'')$ for all $d'' \in \mathcal{C}(\Sigma_{\mathrm{d}})$.

So this concludes the proof of Theorem 8.33. ⊠

**Example 8.36** Consider the transition system specification of Example 8.4. Obviously the deduction rules are in the process-tyft format. They also satisfy the sfisl format as no variables introduced in the target of any premise are used in the source of a premise or in the target of the conclusion. Variable $x_1$ in deduction rule (5) is unresolved. Hence, we obtain $IV_f \supseteq \{1\}$. As the process function $f$ is not used in any other deduction rule we find $IV_f = \{1\}$. Obviously, for all process constants we find that the set $IV$ is empty: $IV_a = IV_b = IV_c = \emptyset$. Hence, the transition system specification is also in sfisl format. From this we conclude that initially stateless bisimilarity is a congruence.

In the next two examples, we show that none of the two constraints of sfisl can be relaxed in any conceivable way.

**Example 8.37** Consider the following transition system specification (with process constants $a$, $b$, $c$, and $c'$, unary process function $f$, and data constants $d$ and $d'$) and the following deduction rules:

$$(1)\ \frac{}{(a, d) \xrightarrow{l} (c, d)} \qquad (2)\ \frac{}{(b, d) \xrightarrow{l} (c', d)}$$

$$(3)\ \frac{}{(c, d') \xrightarrow{l} (c, d')} \qquad (4)\ \frac{(x, v) \xrightarrow{l} (y, d)}{(f(x), v) \xrightarrow{l} (y, d')}$$

The deduction rules (1)-(3) are in the sfisl format, trivially. Deduction rule (4) does not satisfy local data-dependency constraint 1, since $y \in Y_{\mathrm{p}}$ but $d \neq d'$. Local data-dependency constraint 2 and the global data-dependency constraint are satisfied (with $IV_f = \emptyset$).

That initially stateless bisimilarity is not a congruence w.r.t. $f$ can be seen as follows: we have that $a \leftrightarroweq_{isl} b$, but not that $f(a) \leftrightarroweq_{isl} f(b)$ since $(f(a), d)$ can perform a transition to $(c, d')$ while $(f(b), d)$ is forced to perform the same transition to $(c', d')$ and it does not hold that $(c, d') \leftrightarroweq_{sb} (c', d')$.

**Example 8.38** Consider the transition system specification from Example 8.37

with deduction rule (4) replaced by:

$$(4)\quad \frac{(x,v)\xrightarrow{l}(y,v') \quad (y,d')\xrightarrow{l}(y',v'')}{(f(x),v)\xrightarrow{l}(y',v'')}$$

The deduction rules (1)-(3) are in the $\mathsf{sfisl}$ format, trivially. Deduction rule (4) satisfies local data-dependency constraint 1 of $\mathsf{sfisl}$, but local data-dependency constraint 2 is not satisfied as $y \in Y_{\mathrm{p}}$ but $d' \neq v'$. The global data-dependency constraint is satisfied by this transition system specification.

That initially stateless bisimilarity is not a congruence w.r.t. $f$ can be seen as follows: $a \mathrel{\underline{\leftrightarrow}}_{isl} b$ holds, but it does not hold that $f(a) \mathrel{\underline{\leftrightarrow}}_{isl} f(b)$ since $(f(a),d)$ is able to perform an $l$ transition (due to rules (4), (3) and (1)) while $(f(b),d)$ deadlocks.

### 8.3.4 Comparing Congruence Results

When motivating the different notions of bisimilarity, we stated that state-based bisimilarity is considered the weakest (least distinguishing) and least robust notion of bisimilarity with respect to data change. This statement, especially the least robust part, may suggest that if for a transition system specification state-based bisimilairty is a congruence, stateless and initially stateless bisimilarity are trivially congruences, as well. This conjecture can be supported by the standard formats that we gave in this section where the state-based format is the most restrictive and stateless is the most relaxed one. Surprisingly, this conclusion is not entirely true. It turns out that congruence for state-based bisimilarity is indeed stronger than congruence for initially stateless bisimilarity but incomparable to congruence for stateless bisimilarity. A similar incomparability result holds for congruence for initially stateless bisimilarity versus stateless bisimilarity, as well.

The following two examples show that congruence results for state-based bisimilarity and stateless bisimilarity are incomparable. In other words, there are both cases in which one of the two notions is a congruence and the other is not.

**Example 8.39** Consider the following transition system specification (with process constants $a$ and $b$, unary process function $f$, and data constants $d$ and $d'$) and the following deduction rules:

$$(1)\quad \frac{}{(a,d')\xrightarrow{l}(a,d')} \qquad (2)\quad \frac{}{(f(a),d)\xrightarrow{l}(a,d')}$$

In the above transition system specification, the process constants $a$ and $b$ are not stateless bisimilar and hence, congruence of stateless bisimilarity follows trivially. However, we have $(a,d) \mathrel{\underline{\leftrightarrow}}_{sb} (b,d)$, but not $(f(a),d) \mathrel{\underline{\leftrightarrow}}_{sb} (f(b),d)$. Hence, congruence of state-based bisimilarity does not hold.

**Example 8.40** Consider the following transition system specification (with process constants $a$, $b$, and $c$, unary process function $f$, and data constants $d$ and $d'$) and the following deduction rules:

$$(1)\ \frac{}{(c,d') \xrightarrow{l'} (c,d')} \qquad (2)\ \frac{}{(f(a),d) \xrightarrow{l} (b,d)}$$

$$(3)\ \frac{}{(f(b),d) \xrightarrow{l} (c,d)} \qquad (4)\ \frac{}{(f(c),d) \xrightarrow{l} (a,d)}$$

State-based bisimilarity is obviously a congruence though the transition system specification does not satisfy the proposed format. Now, consider the processes $a$ and $b$. These two processes are stateless bisimilar, however, $f(a)$ and $f(b)$ are not stateless bisimilar, since $(f(a),d)$ can make a transition to $(b,d)$, whereas $(f(b),d)$ is forced to make a transition to $(c,d)$. Clearly, $b$ and $c$ are not stateless bisimilar (due to their difference w.r.t. data state $d'$).

The following lemma states that if state-based bisimilarity is a congruence, then initially stateless bisimilarity is a congruence as well.

**Lemma 8.41** For a transition system specification, if state-based bisimilarity is a congruence, then initially stateless bisimilarity is a congruence, as well.

*Proof.* Consider an arbitrary $(f,(ar(f)) \in \Sigma_{\mathrm{p}}$ and suppose that for some $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{p}_{ar(f)-1} \in \mathcal{C}(\Sigma_{\mathrm{p}})$, $\overrightarrow{p}_{ar(f)-1} \leftrightarrows_{isl} \overrightarrow{q}_{ar(f)-1}$. By definition this means that there exist state-based bisimulation relations $R_i$ such that $((p_i,d),(q_i,d)) \in R_i$ for all $d$. Since state-based bisimilarity is a congruence (by assumption), we have, for each $d$, the existence of a state-based bisimulation relation $S_d$ such that $(f(\overrightarrow{p}_{ar(f)-1}),d)$ $S_d$ $(f(\overrightarrow{q}_{ar(f)-1}),d)$. Let $S = \bigcup_d S_d$, and observe that $S$ is a state-based bisimulation relation such that, for all $d$, $(f(\overrightarrow{p}_{ar(f)-1}),d)$ $S$ $(f(q_0,\ldots,q_{n-1}),d)$. This means that $f(\overrightarrow{p}_{ar(f)-1})$ $\leftrightarrows_{isl}$ $f(\overrightarrow{q}_{ar(f)-1})$. ⊠

**Corollary 8.42** If a transition system specification is in the sfsb format, then initially stateless bisimilarity is a congruence for it.

Lemma 8.41 shows that congruence for initially stateless bisimilarity is either stronger than or incomparable to congruence for stateless bisimilarity (since in Example 8.40, we have already shown that there exists a case were state-based bisimilarity, thus initially stateless bisimilarity, is a congruence but stateless bisimilarity is not). To prove the incomparability result, we need a counterexample where stateless bisimilarity is a congruence but initially stateless bisimilarity is not (the counterexample of Example 8.39 does not work in this case). The following example establishes this fact.

**Example 8.43** Consider the following transition system specification (with process constants $a$, $b$, and $c$, unary process function $f$, and data constants $d$ and $d'$) and the following deduction rules:

$$(1)\ \frac{}{(a,d') \xrightarrow{l} (a,d)} \qquad (2)\ \frac{}{(b,d') \xrightarrow{l} (c,d')} \qquad (3)\ \frac{}{(c,d) \xrightarrow{l} (c,d)}$$

$$(4)\ \frac{}{(f(a),d) \xrightarrow{l} (c,d)} \qquad (5)\ \frac{}{(f(b),d') \xrightarrow{l} (c,d')}$$

According to the above transition system specification, none of the three constants $a$, $b$ and $c$ are stateless bisimilar, thus congruence of stateless bisimilarity is obvious. However, we have $a \leftrightarrow_{isl} b$ but not $f(a) \leftrightarrow_{isl} f(b)$.

So, to conclude, we have proved in this section, that congruence for state-based bisimilarity implies congruence for initially stateless bisimilarity (and not vice versa). However, proving congruence for stateless bisimilarity does not necessarily mean anything for congruence for the two other notions.

### 8.3.5 Seasoning the Process-tyft Format

The deduction rules in all three proposed formats are of the following form:

$$\frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') \mid i \in I\}}{(f(x_0, \ldots, x_{n-1}), u) \xrightarrow{l}_{r} (t, u')}.$$

Using this form we cannot go far with proving congruence properties of existing theories since there are many other constructs and patterns that are not present in the above format. In this section, we show how to exploit the formats in presence of such constructs.

#### Deduction Rules in Tyxt Format

A common type of deduction rule used in transition system specifications is the *tyxt* form which has the following structure:

$$(dr)\ \frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') \mid i \in I\}}{(x, u) \xrightarrow{l}_{r} (t', u')}.$$

Rules of the above form fit within the *tyft* form if we replace it with a copy the above rule for each function symbol $(f, ar(f)) \in \Sigma_{\mathrm{p}}$ where all occurrences of $x$ are replaced by $f(\overrightarrow{x}_{ar(f)-1})$ with $x_i \notin vars(dr)$:

$$(dr_f)\ \frac{\{(t_i[f(x_0, \ldots, x_{n-1})/x], u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') | i \in I\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) \xrightarrow{l}_{r} (t'[f(\overrightarrow{x}_{ar(f)-1})/x], u')}.$$

Observe that the resulting deduction rule is indeed in the process-tyft format. In [64], it is shown that the original transition system specification and the unfolded one are transition equivalent (meaning that the same transitions can be derived).

For our congruence results there is no problem in also allowing deduction rules in tyxt format. It is not necessary to explicitly transform the transition system specification as described above to apply our congruence results for stateless and state-based bisimilarity since deduction rules in tyxt format transform into deduction rules in the process-tyft format and since any data-dependency constraint involving $x$ in the original deduction rule is replaced by data-dependency constraints involving the $x_i$ variables in the unfolded deduction rule and vice versa.

Checking whether initially stateless bisimilarity is a congruence is not as straightforward due to the global data-dependency constraint. There are two simple solutions. First, check whether state-based bisimilarity is a congruence; if so, so is initially stateless bisimilarity. Or second, check congruence of initially stateless bisimilarity on the unfolded transition system specification.

### Deduction Rules with Predicates

Another common phenomenon is the presence of predicates. Predicates of the form $P(t, u)$ may be present in the premises or the conclusion of a deduction rule. Thus, we allow deduction rules of the following forms:

$$(dr_1)\quad \frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') | i \in I\} \cup \{P_j(t_j', v_j) \mid j \in J\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) \xrightarrow{l}_{r} (t', u')}$$

and

$$(dr_2)\quad \frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') | i \in I\} \cup \{P_j(t_j', v_j) \mid j \in J\}}{P(f(\overrightarrow{x}_{ar(f)-1}), u)}.$$

Predicates can be dealt with in the above formats, as if they are source of a transition relation. This can be formally proved by introducing a fresh dummy transition relation for each predicate and replacing occurrences of that predicate in premises by this transition relation with a target consisting of a fresh dummy process variable and a fresh dummy data variable and occurrences in conclusions by this transition relation with a state consisting of a fresh process constant and a fresh data constant in the target. This transformation is similar to the transformation from [12] for the path format.

Hence, a deduction rule of the form $(dr_1)$ is replaced by a deduction rule of the form

$$(dr_1')\quad \frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') | i \in I\} \cup \{(t_j', v_j) R_{P_j} (z_j, w_j) \mid j \in J\}}{(f(\overrightarrow{x}_{ar(f)-1})), u) \xrightarrow{l}_{r} (t', u')}$$

where the $z_j$ are all different process variables that did not occur in $(dr_1)$ and the $w_j$ are all different data variables that did not occur in $(dr_1)$.

A deduction rule of the form $(dr_2)$ is replaced by a deduction rule of the form

$$(dr_2') \quad \frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') | i \in I\} \cup \{(t_j', v_j) R_{P_j}(z_j, w_j) \mid j \in J\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) R_P(a, d)}.$$

where additionally $a$ is a process constant such that $a \notin \Sigma_{\mathrm{p}}$ and $d$ is a data constant such that $d \notin \Sigma_{\mathrm{d}}$.

As before, this transformation does not have to be carried out explicitly. Observe that the original transition system specification is in the process-tyft format (by considering the arguments of the predicates as sources of premises and conclusions) iff the transformed transition system specification is in the process-tyft format. Thus, stateless bisimilarity is a congruence for the original transition system specification iff it is a congruence for the transformed transition system specification.

With respect to the sfsb format, observe that the sources of data-dependencies are the same for the original rules and the transformed rules. This is due to the decision to treat the argument of the predicates as sources of conclusions and premises. Also observe that there are new targets of premises introduced by the transformation, but those only contain fresh variables and can therefore never be used to satisfy a data-dependency constraint. Hence, the transformed transition system specification satisfies the sfsb format when the original transition system specification does.

For the local data-dependencies of the sfisl format a similar observation holds. For, the new process constant $a$, we find $IV_a = \emptyset$ and for all other process constants and functions we find that the sets $IV$ are the same for the original transition system specification and the transformed one. Therefore, the transformed transition system specification satisfies the sfisl format iff the original transition system specification does.

### Deduction Rules with Negative Premises

As argued by Groote [61], it is often convenient to describe that certain activity can be performed based on the absence of certain actions. Thus, we allow for deduction rules of the following form:

$$(dr) \quad \frac{\{(t_i, u_i) \xrightarrow{l_i}_{r_i} (y_i, u_i') \mid i \in I\} \cup \{(t_j, u_j) \overset{l_j}{\nrightarrow}_{r_j} \mid j \in J\}}{(f(\overrightarrow{x}_{ar(f)-1}), u) \xrightarrow{l}_r (t', u')}.$$

For such transition system specifications another definition is required of what a proof of a transition is (see [61, 135]). Not every transition system specification with negative premises defines a transition relation. Different interpretations of negative premises can be considered (see [59]), but here we adopt the interpretation put forward by [61]. A sufficient condition for the existence of a transition relation is that the transition system specification is *stratifiable*.

A stratification is a metric on formulae that, for each deduction rule of the transition system specification, does not increase from conclusion to all positive premises and strictly decreases from conclusion to negative premises (i.e., if a stratification for all rules exists). For stratifiable transition system specifications, our congruence results can be used safely.

## 8.4 Applications of the Formats

In this section, some process languages from the literature for which an operational semantics is provided by means of a transition system specification with a data state are considered.

For each of these languages, we establish which of the notions of bisimilarity introduced in this article are used (possibly with a different formulation) and whether the deduction rules are in the corresponding format. We focus on stateless bisimilarity and initially stateless bisimilarity as these seem to be of most interest in these applications.

### 8.4.1 The Coordination Language Linda

In [35], the operational semantics of Linda is given using a combination of SOS rules and a structural congruence. As this kind of transition system specification is not purely in the format used in this article, we have transformed it in such a way that it fits the format (by extending the language with a process constant $\epsilon$). A formulation of this semantics without the constant $\epsilon$ is also possible, but the resulting transition system specification is much larger. In this section, we apply the proposed formats on the extended language. Process constants (atomic process terms) in this language are $\epsilon$ (for terminating process), $ask(t)$ and $nask(t)$ (for checking existence and absence of tuple $t$ in the shared data space, respectively), $tell(t)$ (for adding tuple $t$ to the space) and $get(t)$ (for taking tuple $t$ from the space). Process composition operators in this language include nondeterministic choice ($+$), sequential composition ($;$) and parallel composition ($||$). The data signature of this language consists of a constant $\{\}$ for the empty multiset and a class of unary function symbols $\cup\{t\}$, for all tuples $t$, denoting the union of a multiset with a singleton multiset containing tuple $t$. The operational state of a Linda program is denoted by $(p, \varsigma)$ where $p$ is a process term in the above syntax and $\varsigma$ is a multiset modelling the shared data space.

The transition system specification defines one relation $\rightarrow$ and one predicate $\downarrow$. Negative premises and deduction rules in tyxt format are not used.

The deduction rules of our reformulation of the SOS for Linda from [35] are the following:

$$(1)\ \frac{}{(\epsilon,\varsigma)\downarrow} \qquad (2)\ \frac{}{(ask(t),\varsigma\cup\{t\})\rightarrow(\epsilon,\varsigma\cup\{t\})}$$

$$(3)\ \frac{}{(tell(t),\varsigma)\rightarrow(\epsilon,\varsigma\cup\{t\})} \qquad (4)\ \frac{}{(get(t),\varsigma\cup\{t\})\rightarrow(\epsilon,\varsigma)}$$

$$(5)\ \frac{[t\notin\varsigma]}{(nask(t),\varsigma)\rightarrow(\epsilon,\varsigma)} \qquad (6)\ \frac{(x_0,\varsigma)\downarrow}{(x_0+x_1,\varsigma)\downarrow} \qquad (7)\ \frac{(x_1,\varsigma)\downarrow}{(x_0+x_1,\varsigma)\downarrow}$$

$$(8)\ \frac{(x_0,\varsigma)\rightarrow(y,\varsigma')}{(x_0+x_1,\varsigma)\rightarrow(y,\varsigma')} \qquad (9)\ \frac{(x_1,\varsigma)\rightarrow(y,\varsigma')}{(x_0+x_1,\varsigma)\rightarrow(y,\varsigma')}$$

$$(10)\ \frac{(x_0,\varsigma)\rightarrow(y,\varsigma')}{(x_0\ ;\ x_1,\varsigma)\rightarrow(y\ ;\ x_1,\varsigma')} \qquad (11)\ \frac{(x_0,\varsigma)\downarrow\ \ (x_1,\varsigma)\rightarrow(y,\varsigma')}{(x_0\ ;\ x_1,\varsigma)\rightarrow(y,\varsigma')}$$

$$(12)\ \frac{(x_0,\varsigma)\rightarrow(y,\varsigma')}{(x_0||x_1,\varsigma)\rightarrow(y||x_1,\varsigma')} \qquad (13)\ \frac{(x_1,\varsigma)\rightarrow(y,\varsigma')}{(x_0||x_1,\varsigma)\rightarrow(x_0||y,\varsigma')}$$

$$(14)\ \frac{(x_0,\varsigma)\downarrow\ \ (x_1,\varsigma)\downarrow}{(x_0\ ;\ x_1,\varsigma)\downarrow} \qquad (15)\ \frac{(x_0,\varsigma)\downarrow\ \ (x_1,\varsigma)\downarrow}{(x_0||x_1,\varsigma)\downarrow}$$

Obviously these deduction rules are all in the process-tyft format (with appropriate seasoning for termination predicate $\downarrow$). As a consequence, stateless bisimilarity is a congruence. Initially stateless bisimilarity is a congruence for all operators except parallel composition. Note that $IV_+ = \emptyset$ and $IV_; = \{1\}$. Thus, initially stateless bisimilarity is a congruence for the sequential part of Linda.

Because congruence of initially stateless bisimilarity w.r.t. parallel composition cannot be concluded using our format, we may wonder whether this result must have been expected. In the following example, we show that this is indeed the case and the indications given by our format are true (i.e., initially stateless bisimilarity is not a congruence for the language with parallel composition operator).

**Example 8.44** Consider the processes $p = ask(1)\ ;\ (nask(1)\ ;\ ask(2))$ and $q = ask(1)\ ;\ nask(1)$. According to the above transition system specification, it holds that $p \underline{\leftrightarrow}_{isl}\, q$ (in both processes, using an arbitrary common initial state, either $ask(1)$ executes followed by deadlock or both deadlock immediately). However, if we compose each of the two processes in parallel with the process $r = get(1)$, then the two processes may behave differently for some data states. For example, consider the data state $\{1, 2\}$. For this data state, one execution path of $(p \parallel r, \{1, 2\})$ is: first executing $ask(1)$ from $p$ successfully, then $get(1)$ from $r$ (thus, resulting in data state $\{2\}$), and executing $nask(1)$ followed by $ask(2)$ successfully. However, all possible executions of $(q \parallel r, \{1, 2\})$ can never make four consecutive transitions before termination. Thus, we conclude that initially stateless bisimilarity is not a congruence with respect to the parallel composition.

### 8.4.2 The Timed Process Algebra Timed $\mu$CRL

In [62], a timed extension of the language $\mu$CRL, called timed $\mu$CRL, is defined. In this section, we consider a fragment of this language consisting of the following process constants and functions:

- process constants: $\delta$, $(a)_{a \in A}$;
- unary process functions: $\left(\sum_x \_\right)_{x \in V}$, $(\_ {}^{\varsigma} t)_{t \in T}$;
- binary process functions: $+$, $\cdot$, $(\_ \lhd b \rhd \_)_{b \in B}$, $\parallel$.

The meaning of the sets $A$, $V$, $T$, and $B$, and the meaning of the process constants and functions are irrelevant. The process functions that we do not consider here, are either only introduced for axiomatization purposes ($\lfloor\!\lfloor$, $|$, $\ll$) or renaming of actions ($\partial_H$, $\rho_R$, $\tau_I$). The transition system specification defines the following predicates and relations:

- a 'delay-predicate' $U$;
- a family of 'action-termination' predicates $\left(\_ \xrightarrow{a} \checkmark\right)_{a \in A}$;
- a family of 'action-transition' relations $\left(\_ \xrightarrow{a} \_\right)_{a \in A}$;
- a 'time-transition' relation $\_ \xrightarrow{\iota} \_$.

The data state consists of an element of the set $T$ (reflecting time). In [62], $U(p, t)$ is written as $U(t, p)$ and $(\, p, t\,) \xrightarrow{a} \checkmark$ is written as $(\, p, t\,) \xrightarrow{a} (\, \checkmark, t\,)$. In this section, we use the notations $U(p, t)$ and $(\, p, t\,) \xrightarrow{a} \checkmark$. The deduction rules are given below.

$$(1)\ \frac{}{(\, a, t\,) \xrightarrow{a} \checkmark} \qquad (2)\ \frac{}{U(a, t)} \qquad (3)\ \frac{}{U(\delta, t)}$$

$$(4)\quad \frac{(\,x_0,t\,) \xrightarrow{l} \checkmark}{\begin{array}{c}(\,x_0+x_1,t\,) \xrightarrow{l} \checkmark\\ (\,x_1+x_0,t\,) \xrightarrow{l} \checkmark\end{array}} \qquad (5)\quad \frac{(\,x_0,t\,) \xrightarrow{l} (\,y,t\,)}{\begin{array}{c}(\,x_0+x_1,t\,) \xrightarrow{l} (\,y,t\,)\\ (\,x_1+x_0,t\,) \xrightarrow{l} (\,y,t\,)\end{array}}$$

$$(6)\quad \frac{U(x_0,t)}{\begin{array}{c}U(x_0+x_1,t)\\ U(x_1+x_0,t)\end{array}} \qquad (7)\quad \frac{(\,x_0,t\,) \xrightarrow{l} \checkmark}{(\,x_0\cdot x_1,t\,) \xrightarrow{l} (\,x_1,t\,)}$$

$$(8)\quad \frac{(\,x_0,t\,) \xrightarrow{l} (\,y,t\,)}{(\,x_0\cdot x_1,t\,) \xrightarrow{l} (\,y\cdot x_1,t\,)} \qquad (9)\quad \frac{U(x_0,t)}{U(x_0\cdot x_1,t)}$$

$$(10)\quad \frac{(\,x_0,t\,) \xrightarrow{l} \checkmark \quad [\models b]}{(\,x_0 \lhd b \rhd x_1,t\,) \xrightarrow{l} \checkmark} \qquad (11)\quad \frac{(\,x_1,t\,) \xrightarrow{l} \checkmark \quad [\not\models b]}{(\,x_0 \lhd b \rhd x_1,t\,) \xrightarrow{l} \checkmark}$$

$$(12)\quad \frac{(\,x_0,t\,) \xrightarrow{l} (\,y,t\,) \quad [\models b]}{(\,x_0 \lhd b \rhd x_1,t\,) \xrightarrow{l} (\,y,t\,)} \qquad (13)\quad \frac{(\,x_1,t\,) \xrightarrow{l} (\,y,t\,) \quad [\not\models b]}{(\,x_0 \lhd b \rhd x_1,t\,) \xrightarrow{l} (\,y,t\,)}$$

$$(14)\quad \frac{U(x_0,t) \quad [\models b]}{U(x_0 \lhd b \rhd x_1,t)} \qquad (15)\quad \frac{U(x_1,t) \quad [\not\models b]}{U(x_0 \lhd b \rhd x_1,t)}$$

$$(16)\quad \frac{(\,x[e/v],t\,) \xrightarrow{l} \checkmark}{\left(\sum_v x,t\right) \xrightarrow{l} \checkmark} \qquad (17)\quad \frac{(\,x[e/v],t\,) \xrightarrow{l} (\,y,t\,)}{\left(\sum_v x,t\right) \xrightarrow{l} (\,y,t\,)} \qquad (18)\quad \frac{U(x[e/v],t)}{U(\sum_v x,t)}$$

$$(19)\quad \frac{(\,x,t\,) \xrightarrow{l} \checkmark}{(\,x\,{}^{\varsigma}t,t\,) \xrightarrow{l} \checkmark} \qquad (20)\quad \frac{(\,x,t\,) \xrightarrow{l} (\,y,t\,)}{(\,x\,{}^{\varsigma}t,t\,) \xrightarrow{l} (\,y,t\,)} \qquad (21)\quad \frac{U(x,t) \quad [t \le t']}{U(x\,{}^{\varsigma}t',t)}$$

$$(22)\quad \frac{U(x,t') \quad [t < t']}{(\,x,t\,) \xrightarrow{\iota} (\,x,t'\,)}$$

$$(23)\quad \frac{(\,x_0,t\,) \xrightarrow{l} \checkmark \quad (\,x_1,t\,) \xrightarrow{l'} \checkmark \quad [\gamma(l,l') = l'']}{(\,x_0 \parallel x_1,t\,) \xrightarrow{l''} \checkmark}$$

$$(24)\quad \frac{(\,x_0,t\,) \xrightarrow{l} \checkmark}{\begin{array}{c}(\,x_0 \parallel x_1,t\,) \xrightarrow{l} (\,x_1,t\,)\\ (\,x_1 \parallel x_0,t\,) \xrightarrow{l} (\,x_1,t\,)\end{array}} \qquad (25)\quad \frac{(\,x_0,t\,) \xrightarrow{l} (\,y,t\,)}{\begin{array}{c}(\,x_0 \parallel x_1,t\,) \xrightarrow{l} (\,y \parallel x_1,t\,)\\ (\,x_1 \parallel x_0,t\,) \xrightarrow{l} (\,x_1 \parallel y,t\,)\end{array}}$$

$$(26)\quad \frac{(\,x_0, t\,) \xrightarrow{l} \checkmark \quad (\,x_1, t\,) \xrightarrow{l'} (\,y, t\,) \quad [\gamma(l, l') = l'']}{(\,x_0 \,\|\, x_1, t\,) \xrightarrow{l''} (\,y, t\,) \quad (\,x_1 \,\|\, x_0, t\,) \xrightarrow{l''} (\,y, t\,)}$$

$$(27)\quad \frac{(\,x_0, t\,) \xrightarrow{l} (\,y_0, t\,) \quad (\,x_1, t\,) \xrightarrow{l'} (\,y_1, t\,) \quad [\gamma(l, l') = l'']}{(\,x_0 \,\|\, x_1, t\,) \xrightarrow{l''} (\,y_0 \,\|\, y_1, t\,)}$$

$$(28)\quad \frac{U(x_0, t) \quad U(x_1, t)}{U(x_0 \,\|\, x_1, t)}$$

Observe that in this transition system specification two relations and two predicates are used. Negative premises do not occur, but there is a deduction rule in tyxt format.

The equivalence used in [62] for timed $\mu$CRL process terms is timed bisimilarity, which coincides with our notion of initially stateless bisimilarity. The definition of timed bisimilarity in [62] does not require the delay predicate to be transferred between related processes. The notion of initially stateless bisimilarity presented in this article is based on transferring all predicates and relations used in the transition system specification. This difference is not problematic as it can easily be proved that any two timed bisimilar process terms are also initially stateless bisimilar and vice versa. Congruence of timed bisimilarity is claimed without proof in [62]. In [109], a reformulation of the semantics of timed $\mu$CRL is given in such a way that the data state is encoded into the process terms at the expense of an auxiliary operator. Then, the notion of timed bisimilarity corresponds with the traditional notion of bisimilarity, for which congruence is proven using traditional means.

**Stateless bisimilarity** Note that although stateless bisimilarity is not considered in [62], from the format of the deduction rules, congruence for this equivalence follows easily.

**State-based bisimilarity** All deduction rules of timed $\mu$CRL are in the sfsb format except for deduction rule (22). For this deduction rule, the data-dependency constraints 1 and 3 of sfsb are violated in the target of the conclusion and the source of the (only) premise as $x \in X_\mathrm{p}$ but $t' \neq t$ (note that data-dependency constraints 2 and 4 are respected by this rule). Hence, state-based bisimilarity cannot be concluded to be a congruence for any of the non-nullary[1] process functions of timed $\mu$CRL.

[1] For nullary process functions there is no data dependency at all.

Nevertheless, using traditional means one can quite easily establish that state-based bisimilarity is a congruence for some of the operators, for example alternative composition.

**Initially stateless bisimilarity** Before we discuss initially stateless bisimilarity in more detail we emphasize that deduction rule (22) needs to be transformed before the format can be applied. Deduction rule (22) maps to a collection of deduction rules of the form

$$(22_f) \quad \frac{U(f(x_0,\ldots,x_{n-1}),t') \quad [t < t']}{(\, f(x_0,\ldots,x_{n-1}),t\,) \xrightarrow{\iota} (\, f(x_0,\ldots,x_{n-1}),t'\,)};$$

one for each $n$-ary process function $f$ in the signature of timed $\mu$CRL.

All deduction rules of timed $\mu$CRL except for deduction rules derived from deduction rule (22) for non-nullary process functions are in the sfsb format. Thus, with respect to the local constraints of sfisl, only those derived deduction rules have to be considered.

Note that the set of variables $Y_\mathrm{p}$ is empty for such a deduction rule. Hence, the local data-dependency constraints of sfisl are satisfied trivially.

For an arbitrary function symbol $f$ with arity $n$, the set of unresolved variables consists of the indices of all arguments. As a consequence, $IV_f \supseteq \{0,\ldots,n-1\}$. For all process functions, except for sequential and parallel composition, the defining deduction rules do not contain any occurrences of process functions in the source of a premise or in the target of the conclusion. Hence, for all those process functions, we obtain $IV_f$ is the set of all indices of $f$.

For sequential composition (deduction rule (8)) and parallel composition (deduction rules (25) and (27)) the occurrences of $y$, $y_0$ and $y_1$ in the use of the process functions do not satisfy the requirement that these should be initial variables ($\in X_\mathrm{p}$). Hence, for those process functions, the set $IV$ does not exist. Therefore, congruence of initially stateless bisimilarity w.r.t. those process functions cannot be concluded. For the other process functions, as they are independently defined operationally, congruence can be concluded.

We claim that a reformulation of the operational semantics of timed $\mu$CRL without the predicate $U$ along the following lines results in an 'equivalent' transition system specification for which the sfisl format can be applied to obtain congruence:

$$\frac{(\, x_0,t\,) \xrightarrow{\iota} (\, y,t'\,)}{(\, x_0 \cdot x_1,t\,) \xrightarrow{\iota} (\, y \cdot x_1,t\,)}, \qquad \frac{(\, x_0,t\,) \xrightarrow{\iota} (\, y_0,t'\,) \quad (\, x_1,t\,) \xrightarrow{\iota} (\, y_1,t'\,)}{(\, x_0 \,\|\, x_1,t\,) \xrightarrow{\iota} (\, y_0 \,\|\, y_1,t'\,)}.$$

The reason is that the first argument of sequential composition and both arguments of parallel composition are no longer forced to be part of the set $IV$ which avoids the problem with $y$, $y_0$ and $y_1$ not being initial variables. Calculation of the sets $IV_{\cdot}$ and $IV_{\parallel}$ gives: $IV_{\cdot} = \emptyset$ and $IV_{\parallel} = \emptyset$.

### 8.4.3 The Hybrid Process Algebra HyPA

In [41], a process algebra is presented for the description of hybrid systems, i.e., systems with both discrete events and continuous change of variables. The process signature of HyPA consists of the following process constants and functions:

- process constants: $\delta$, $\epsilon$, $(a)_{a\in A}$, $(c)_{c\in C}$;
- unary process functions: $(d \gg \_)_{d\in D}$, $(\partial_H(\_))_{H\subseteq A}$;
- binary process functions: $\oplus$, $\odot$, $\blacktriangleright$, $\triangleright$, $\parallel$, $\lfloor\!\lfloor$, and $|$.

Negative premises and rules in tyxt format are not used in this transition system specification.

We refrain from giving further information about the intended meaning of the sets $A$, $C$, and $D$, and the meaning of the process constants and functions as these are irrelevant to the application of our congruence theorems on this language. The data state consists of mappings from model variables to values, denoted by $Val$. The data signature is not made explicit.

The transition system specification defines the following predicate and relations:

- a 'termination'-predicate $\checkmark$;
- a family of 'action-transition' relations $\left(\_ \xrightarrow{l} \_\right)_{l\in A\times Val}$;
- a family of 'flow-transition' relations $\left(\_ \overset{\sigma}{\rightsquigarrow} \_\right)_{\sigma\in T\to Val}$.

Also, the meaning of the set $T$ is irrelevant for our purposes. The deduction rules are given below.

$$(1)\ \frac{}{(\,\epsilon,\nu\,)\checkmark} \qquad (2)\ \frac{}{(\,a,\nu\,)\xrightarrow{a,\nu}(\,\epsilon,\nu\,)} \qquad (3)\ \frac{[(\nu,\sigma)\models_{\mathrm{f}} c]\quad [dom(\sigma)=[0,t]]}{(\,c,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,c,\sigma(t)\,)}$$

$$(4)\ \frac{[(\nu,\nu')\models_{\mathrm{r}} d]\quad (\,x,\nu'\,)\checkmark}{(\,d\gg x,\nu\,)\checkmark} \qquad (5)\ \frac{[(\nu,\nu')\models_{\mathrm{r}} d]\quad (\,x,\nu'\,)\xrightarrow{l}(\,y,\nu''\,)}{(\,d\gg x,\nu\,)\xrightarrow{l}(\,y,\nu''\,)}$$

$$(6)\quad \frac{(\,x_0,\nu\,)\checkmark}{\begin{array}{c}(\,x_0 \oplus x_1,\nu\,)\checkmark\\ (\,x_1 \oplus x_0,\nu\,)\checkmark\end{array}} \qquad (7)\quad \frac{(\,x_0,\nu\,)\xrightarrow{l}(\,y,\nu'\,)}{\begin{array}{c}(\,x_0 \oplus x_1,\nu\,)\xrightarrow{l}(\,y,\nu'\,)\\ (\,x_1 \oplus x_0,\nu\,)\xrightarrow{l}(\,y,\nu'\,)\end{array}}$$

$$(8)\quad \frac{(\,x_0,\nu\,)\checkmark \quad (\,y_0,\nu\,)\checkmark}{(\,x_0 \odot y_0,\nu\,)\checkmark} \qquad (9)\quad \frac{(\,x_0,\nu\,)\xrightarrow{l}(\,y,\nu'\,)}{(\,x_0 \odot x_1,\nu\,)\xrightarrow{l}(\,y \odot x_1,\nu'\,)}$$

$$(10)\quad \frac{(\,x_0,\nu\,)\checkmark \quad s\,(\,x_1,\nu\,)\xrightarrow{l}(\,y,\nu'\,)}{(\,x_0 \odot x_1,\nu\,)\xrightarrow{l}(\,y,\nu'\,)}$$

$$(11)\quad \frac{(\,x_0,\nu\,)\checkmark}{\begin{array}{c}(\,x_0 \blacktriangleright x_1,\nu\,)\checkmark\\ (\,x_0 \rhd x_1,\nu\,)\checkmark\end{array}} \qquad (12)\quad \frac{(\,x_0,\nu\,)\xrightarrow{l}(\,y,\nu'\,)}{\begin{array}{c}(\,x_0 \blacktriangleright x_1,\nu\,)\xrightarrow{l}(\,y \blacktriangleright x_1,\nu'\,)\\ (\,x_0 \rhd x_1,\nu\,)\xrightarrow{l}(\,y \blacktriangleright x_1,\nu'\,)\end{array}}$$

$$(13)\quad \frac{(\,x_1,\nu\,)\checkmark}{(\,x_0 \blacktriangleright x_1,\nu\,)\checkmark} \qquad (14)\quad \frac{(\,x_1,\nu\,)\xrightarrow{l}(\,y,\nu'\,)}{(\,x_0 \blacktriangleright x_1,\nu\,)\xrightarrow{l}(\,y,\nu'\,)}$$

$$(15)\quad \frac{(\,x_0,\nu\,)\checkmark \quad (\,x_1,\nu\,)\checkmark}{(\,x_0 \parallel x_1,\nu\,)\checkmark\,(\,x_0 \mid x_1,\nu\,)\checkmark} \qquad (16)\quad \frac{\begin{array}{c}(\,x_0,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y_0,\nu'\,)\\ (\,x_1,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y_1,\nu'\,)\end{array}}{\begin{array}{c}(\,x_0 \parallel x_1,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y_0 \parallel y_1,\nu'\,)\\ (\,x_0 \mid x_1,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y_0 \parallel y_1,\nu'\,)\end{array}}$$

$$(17)\quad \frac{(\,x_0,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y,\nu'\,) \quad (\,x_1,\nu\,)\checkmark}{\begin{array}{c}(\,x_0 \parallel x_1,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y,\nu'\,)\\ (\,x_1 \parallel x_0,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y,\nu'\,)\\ (\,x_0 \mid x_1,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y,\nu'\,)\\ (\,x_1 \mid x_0,\nu\,)\overset{\sigma}{\rightsquigarrow}(\,y,\nu'\,)\end{array}} \qquad (18)\quad \frac{(\,x_0,\nu\,)\xrightarrow{a,\nu'}(\,y,\nu''\,)}{\begin{array}{c}(\,x_0 \parallel x_1,\nu\,)\xrightarrow{a,\nu'}(\,y \parallel x_1,\nu''\,)\\ (\,x_1 \parallel x_0,\nu\,)\xrightarrow{a,\nu'}(\,x_1 \parallel y,\nu''\,)\\ (\,x_0 \mathbin{\|\!\lfloor} x_1,\nu\,)\xrightarrow{a,\nu'}(\,y \parallel x_1,\nu''\,)\end{array}}$$

$$(19)\quad \frac{(\,x_0,\nu\,)\xrightarrow{a,\nu'}(\,y_0,\nu''\,) \quad (\,x_1,\nu\,)\xrightarrow{a',\nu'}(\,y_1,\nu''\,) \quad [a'' = a\,\gamma\,a']}{\begin{array}{c}(\,x_0 \parallel x_1,\nu\,)\xrightarrow{a'',\nu'}(\,y_0 \parallel y_1,\nu''\,)\\ (\,x_0 \mid x_1,\nu\,)\xrightarrow{a'',\nu'}(\,y_0 \parallel y_1,\nu''\,)\end{array}}$$

$$(20)\quad \frac{(\,x,\nu\,)\xrightarrow{a,\nu'}(\,y,\nu''\,) \quad [a \notin H]}{(\,\partial_H\,(x)\,,\nu\,)\xrightarrow{a,\nu'}(\,\partial_H\,(y)\,,\nu''\,)}$$

$$(21)\ \frac{(x,\nu) \overset{\sigma}{\rightsquigarrow} (y,\nu')}{(\partial_H(x),\nu) \overset{\sigma}{\rightsquigarrow} (\partial_H(y),\nu')} \qquad (22)\ \frac{(x,\nu)\ \checkmark}{(\partial_H(x),\nu)\ \checkmark}$$

On HyPA process terms, in [41], a notion of robust bisimilarity is defined that, for HyPA, coincides with our definition of stateless bisimilarity. Furthermore, in [41], for the purpose of analyzing sequential HyPA processes (i.e., HyPA processes without operators for parallel composition), a notion of bisimilarity is defined that coincides with our notion of initially stateless bisimilarity.

**Stateless bisimilarity** One can easily observe that all deduction rules of HyPA are in the process-tyft format. Hence, stateless bisimilarity is a congruence for all constant and function symbols from the process signature of HyPA.

**State-based bisimilarity** With respect to the notion of state-based bisimilarity, as defined in this article, it can be established that state-based bisimilarity is a process-congruence for the constants of HyPA, the alternative composition operator ( $\oplus$ ), and the encapsulation operator ($\partial_H()$), based on the format of the deduction rules. For the other operators however, this is not the case. Deduction rules (4) (after a transformation) and (5) violate data-dependency constraint 3.

Deduction rules (9), (12), and (18) for sequential composition ( $\odot$ ), disrupt ( $\blacktriangleright$ ) and left-disrupt ( $\triangleright$ ), and the parallel composition operators ( $\|$ , $\lfloor\!\lfloor$ , and $|$ ) all violate data-dependency constraint 1 as the data dependency for variable $y$ in the target of the conclusion has no base in the source of the conclusion.

One might wonder whether this means that our format for state-based bisimilarity is too restrictive in the sense that process-congruence cannot be concluded for many operators. This is not the case, for none of these operators state-based bisimilarity is a process-congruence!

**Initially stateless bisimilarity** In case we consider initially stateless bisimilarity, it turns out that the deduction rules are all in sfisl. Hence, what remains is to check whether the global constraints are satisfied. For this, we need to compute the sets $IV_f$ for each process function $f$ of HyPA. For alternative composition and encapsulation, we obtain $IV_{\oplus} = IV_{\partial_H()} = \emptyset$ as there are no unresolved variables in the deduction rules defining these process functions and there are no process functions used in sources of premises or targets of conclusions.

For re-initialization, due to the unresolvedness of variable $x$ (at position 0) in deduction rules (4) and (5), and the fact that no process functions are used in sources of premises or targets of conclusions of re-initialization defining deduction rules, we have $IV_{d\gg} = \{0\}$.

For sequential composition $IV_{\odot} \supseteq \{1\}$ since $x_1$ is unresolved in deduction rule (9). Also note that in the same deduction rule sequential composition is used in the target of the conclusion. The term occurring as argument 1 of this use, $x_1$, is the index 1 variable from the source of the conclusion and hence this occurrence of sequential composition does not add to the set $IV_{\odot}$. As there are no other process functions used in $\odot$-defining deduction rules, we have $IV_{\odot} = \{1\}$. Using a similar reasoning as for sequential composition, we obtain $IV_{\blacktriangleright} = IV_{\triangleright} = \{1\}$.

For the parallel composition operators, based on the unresolvedness of variables in deduction rule (18) we need $IV_{\parallel} \supseteq \{0,1\}$ and $IV_{\lfloor\!\lfloor} \supseteq \{1\}$. All parallel composition operators use parallel composition in the target of at least one of their defining deduction rules. This leads to the additional requirement that all variables occurring in the use of parallel composition are from the set $X_{\mathrm{p}}$. That this is not the case can be seen easily by considering the deduction rules (18) and (19). Hence, it turns out that the sets $IV_{\parallel}$, $IV_{\lfloor\!\lfloor}$, and $IV_{|}$ are not defined.

The transition system specification though does not respect the global constraints imposed by sfisl. However, if we restrict to the part of HyPA without parallel composition operators, i.e., sequential HyPA, then we can conclude that initially stateless bisimilarity is a congruence. In fact, in [41], our congruence theorem for initially stateless bisimilarity has been used to obtain this result.

The fact that we cannot derive that initially stateless bisimilarity is a congruence w.r.t. the parallel composition operators is not a weakness of our format. Also in this case, initially stateless bisimilarity is not a congruence w.r.t. parallel composition. An example of process terms illustrating this for parallel composition is given in [41].

### 8.4.4 The Discrete-event Process Language $\chi_\sigma$

In [29], the process language $\chi_\sigma$ is presented. This language is used for the specification, simulation and validation of discrete-event systems.

The signature of $\chi_\sigma$ consists of the following process constant and function symbols:

- process constants: $\delta$, $\epsilon$, skip, $(\Delta_t)_{t\in T}$, $(x := e)_{x\in V, e\in E}$, $(c!e)_{c\in C, e\in E}$, $(c?x)_{c\in C, x\in V}$;
- unary process functions: $(b \to \_)_{b\in B}$, $\_^*$, $(|[s \mid \_]|)_{s\in S}$, $(\partial_H)_{H\subseteq A}$, $\pi$, $(\tau_I)_{I\subseteq A}$
- binary process functions: $[\!]$, ;, $\parallel$

In the transition system specification of this language both predicates and relations are used. For both types of formulae negative occurrences as a premise occur.

The notion of equivalence that is considered in [29] is (a different formulation of) stateless bisimilarity. The authors attempt to prove that this equivalence is a congruence for the constant and function symbols of $\chi_\sigma$ by using the so-called relaxed PANTH format [83, 9]. For that purpose they consider the begin and end data state of a transition as part of the label of that transition. This way their transition relations and predicates are defined on process terms (without data state). A mistake they make is that in defining which formulae are negative formulae they do not consider the start state as a part of the label. This means that their negative formulae and the ones allowed by [83] are different. Therefore we have serious doubts as to the applicability of the relaxed PANTH format to the given transition system specification of $\chi_\sigma$. Nevertheless, stateless bisimilarity is a congruence since all deduction rules of the transition system specification are in the process-tyft format.

## 8.5 Conclusions

In this chapter, we investigated the impact of the presence of a data state on notions of bisimilarity and standard congruence formats. To do this, we defined three notions of bisimilarity with data and elaborated on their existing and possible uses. Then, we proposed three standard formats that provide congruence results for these three notions. Furthermore, we briefly pointed out the relationships between these notions and between the corresponding congruences. The proposed formats are applied to several examples from the literature successfully. In this article, we illustrated the use of our format using a data coordination language, called Linda, and several process algebras.

Extending the format for a parameterized notion of bisimilarity (with an explicit interference relation or a symbolic/logical representation of interference possibilities) is another interesting extension which should follow the same line as our relaxation of state-based constraints to initially stateless. Furthermore, we may extend the theory to bisimulation relations which allow for different data states but so far we have seen no practical application of such a bisimilarity notion. Investigating the possibility of applying the same techniques for congruence with respect to weaker notions of bisimulation (e.g., branching bisimulation) is another interesting direction for our future research.

We are currently investigating a bi-algebraic and categorical interpretation of notions of bisimulation with data, following the approach of [122, 123, 111]. In this article, we have only proved sufficient conditions for the notions of bisimulation with data to be a congruence. Although we have already shown that no straightforward relaxation of our formats is possible, we could not prove that no relaxation is possible at all. Using the abstract interpretation of semantic rules (as distributive laws), bisimulation and congruence in a co-algebraic settings, we might be able to investigate whether our imposed formats are indeed necessary for congruence or they can be relaxed in any way.

Generating equational theories from transition systems specifications is another direction of our ongoing research. Deriving algebraic axioms for SOS rules in [3, 16] are among notable examples in this direction which try to generate a set of sound and (ground-)complete axioms for a given operational semantics in a syntactic format. Both [3] and [16] assume the existence of a number of standard constants and operators in the signature and we believe that these restrictions on the semantics can be relaxed in several ways (even in a setting without data).

# Chapter 9

# Higher Order Processes

M.C. Escher's "Print Gallery (Rework by Lenstra and de Smit)"


A summarized version of this chapter has appeared as: M.R. Mousavi, M.J. Gabbay, M.A. Reniers, SOS for Higher Order Processes, In M. Abadi and L. de Alfaro eds., *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, San Francisco, CA, USA, volume 3653 of Lecture Notes in Computer Science, pp. 308-322, Springer-Verlag, August 2005.

## 9.1 Introduction

Congruence meta-theorems [5] form an important class of semantic meta-theorems formulated for languages with Structural Operational Semantics. For languages with a higher order notion of behavior (which may emit and receive their own terms as labels), a few proposals exist in the literature [21, 70, 114]. This work's most direct inspiration is from Bernstein's promoted tyft/tyxt format [21] which aims at proving congruence of strong bisimilarity for higher order processes. We lay the foundations for an SOS framework for higher order languages and extend Bernstein's promoted tyft/tyxt, making it both easier to use and strictly more expressive.

For processes with a higher-order behavior, strong bisimilarity might be too restrictive since it requires the emitted or received processes (shown as labels) to be syntactically the same. In practice, however, processes are considered important up to their behavior and hence they should be related using a behavioral (and not syntactic) notion of equality. This leads to a *higher order notion of bisimilarity* [6, 30, 119]. In this chapter, we also present a novel format that is shown to induce congruence for higher order bisimilarity.

*This chapter is organized as follows:* In the next section, we give more details of our contribution in the context of the literature. Section 9.3 fixes the definitions to be used in this chapter. Based on these concepts, our promoted PANTH format is presented in Section 9.4. Section 9.5 studies a higher order notion of bisimilarity and proposes higher order PANTH which induces congruence for this notion. We conclude the chapter and comment on future work in Section 9.6. The SOS framework used in this chapter is a single-sorted TSS with terms as labels as specified in Definition 2.3.

## 9.2 Related Work

**Promoted Tyft/tyxt.** Bernstein in [21] proposes the promoted tyft/tyxt format which extends the tyft/tyxt format by allowing for the use of terms as labels. Rules in this format have the following form:

$$\frac{\{t_i \xrightarrow{t'_i} y_i \mid i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{g(\overrightarrow{z}_{ar(g)-1})} t} \qquad \frac{\{t_i \xrightarrow{t'_i} y_i \mid i \in I\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{z} t}$$

$$\frac{\{t_i \xrightarrow{t'_i} y_i \mid i \in I\}}{x \xrightarrow{g(\overrightarrow{z}_{ar(g)-1})} t} \qquad \frac{\{t_i \xrightarrow{t'_i} y_i \mid i \in I\}}{x \xrightarrow{z} t}$$

The intuition behind the symbols in common with the tyft/tyxt format (Definition 3.6) remains unchanged. For the rest, $g$ is a function symbol, $z_k$'s and $z$ are variables, variables in the source and label of the conclusion and targets of the premises are all distinct and furthermore, all labels of premises are assumed to contain at least one function symbol, i.e., they are not variables. Bernstein proves congruence of strong bisimilarity for TSS's conforming to the promoted tyft/tyxt format.

**Promoted PANTH.** In this chapter, we show that most of the restrictions on labels imposed above are not necessary in general and propose a more general and relaxed format based on the promoted tyft/tyxt format of [21]. We call our new format for strong bisimilarity promoted PANTH. Furthermore, the promoted PANTH format extends syntactic capabilities of the promoted tyft/tyxt format by allowing for predicates, negative premises and lists of terms as labels. We show that the promoted PANTH format is strictly more expressive than promoted tyft/tyxt and point out some usual patterns of SOS rules that the promoted tyft/tyxt format cannot deal with and the promoted PANTH can.

**Proof Methods for Evaluation Systems.** The proof method of Howe [70] and related methods such as those proposed in [113] have been used for proving congruence of applicative bisimulation for functional languages. Sangiorgi also proposes a similar framework in [114] for concurrent extensions of lambda-calculi. Although some of the standard concepts of Howe's method, such as abstraction and evaluation structures, are not explicitly present in our framework, as shown by [21], we can still model the systems studied by [70, 113, 114] and obtain similar results using our formats.

**Higher Order Bisimulation and higher order PANTH.** It was first noted in [6, 30] that there is a need for a notion of behavioral equivalence that relates the behavior of labels instead of their syntax. This notion was also used in [118, 119] for the Calculus of Higher Order Communicating Systems (CHOCS).

In this chapter, we give a general framework for defining the semantics of such systems and proving congruence for the higher order notion of bisimilarity. We also specify CHOCS [119] in our framework, show that the higher order bisimilarity of [119] trivially coincides with ours and conclude that bisimilarity in this framework is indeed a congruence. This way, one can save pages of proof (such as those given explicitly in [119]) for proving congruence.

In [115], it is argued that the higher order notion of bisimilarity may be still too strong for systems with static restriction while it works fine with dynamic restriction of names. It goes beyond the scope of this chapter to discuss this issue but the techniques developed here can be useful in formulating congruence meta-theorems for other notions of bisimilarity for higher order processes (e.g., normal and context bisimilarities of [115]).

It is worth mentioning that in [21], promoted tyft/tyxt is used to prove that higher order bisimilarity is a congruence for CHOCS. But to do so, the semantics of CHOCS is translated into a new semantics and, with a rather lengthy proof, it is shown that higher order bisimilarity in CHOCS coincides with strong bisimilarity in the new semantics. Using our approach, one can save these laborious intermediate steps and arrive at the desired result directly.

**Other SOS Frameworks.** Our SOS framework is closest to that of [48] (simplified by omitting the binding signatures) for which no known congruence format exists. The generalized PANTH format [84] includes variable binding operators (which are not addressed in this chapter), but does not allow for terms as labels and hence cannot deal with higher order process algebras such as CHOCS directly. Galpin in [52] defines a multi-sorted SOS framework with terms as labels. However, there the sort of labels is necessarily different from the sort of processes. Thus, higher-order behavior and higher-order bisimilarity do not have a natural presentation in the extended TSS format of [52].

## 9.3 Preliminaries

We use the TSS framework of Definition 2.3 with single-sorted signatures throughout this chapter. To give an idea of the kind of systems that we are aiming at, we give the TSS of a higher order process algebra called CHOCS [119] which serves as a running example throughout the rest of the chapter.

**Example 9.1** (Calculus of Higher Order Communicating Systems (CHOCS)) The signature of CHOCS consists of the following operators: $0$, $a$, $\tau.\_$, $c!\_.\_$, $c?a.\_$, $\_ \mid\mid \_$, $\_ + \_$, $\_ \setminus c$ and $\_[S]$ where $c$ is taken from the set $C$ of *channel names*, $a$ from the set $A$ of *atoms* and $S : C \to C$ is a function on channel names. (In [119], atoms are called process variables. To avoid confusion with variables in our SOS setting, we use the term *atom* instead.)

Process $0$ is a deadlocking process. An atom $a$ is supposed to represent a "hole" in the process description which can be substituted by another process term. Other than being substituted by a term, an atom does not have any other observable behavior. Internal action prefixing $\tau.p$ first performs a $\tau$-step and then behaves as $p$. A send prefixed process $c!p.p'$ sends process $p$ along the channel $c$ and becomes $p'$ afterwards. A receive prefixed process $c?a.p$, receives a process along $c$ and substitutes it for atom $a$ in $p$. Choice is denoted by $+$ and parallel composition by $\mid\mid$. To make a channel name $c$ internal to process $p$ the restriction expression $p \setminus c$ is used. Finally, the renaming expression $p[S]$ renames all channel names of $p$ as specified by the renaming function $S$.

The transition relations for this formalism are classes of unary substitution $\xrightarrow{t}_{/a}$,

$$\frac{}{a \stackrel{z}{\rightarrow}_{/a} z} \qquad \frac{}{b \stackrel{z}{\rightarrow}_{/a} b}\, a \neq b \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{/a} y_0 \quad x_1 \stackrel{z}{\rightarrow}_{/a} y_1}{c!x_0.x_1 \stackrel{z}{\rightarrow}_{/a} c!y_0.y_1} \qquad \frac{x \stackrel{z}{\rightarrow}_{/b} y}{c?a.x \stackrel{z}{\rightarrow}_{/b} c?a.y}\, a \neq b$$

$$\frac{x_0 \stackrel{z}{\rightarrow}_{/a} y_0 \quad x_1 \stackrel{z}{\rightarrow}_{/a} y_1}{x_0 + x_1 \stackrel{z}{\rightarrow}_{/a} y_0 + y_1} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{/a} y_0 \quad x_1 \stackrel{z}{\rightarrow}_{/a} y_1}{x_0 \mid\mid x_1 \stackrel{z}{\rightarrow}_{/a} y_0 \mid\mid y_1} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{/a} y_0}{x_0 \setminus c \stackrel{z}{\rightarrow}_{/a} y_0 \setminus c} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{/a} y_0}{x_0[S] \stackrel{z}{\rightarrow}_{/a} y_0[S]}$$

$$\frac{}{\tau.x \rightarrow_{\tau} x} \qquad \frac{}{c!x_0.x_1 \stackrel{x_0}{\rightarrow}_{c!} x_1} \qquad \frac{x_1 \stackrel{z}{\rightarrow}_{/a} y_1}{c?a.x_1 \stackrel{z}{\rightarrow}_{c?} y_1}$$

$$\frac{x_0 \rightarrow_{\tau} y_0}{x_0 + x_1 \rightarrow_{\tau} y_0} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c!} y_0}{x_0 + x_1 \stackrel{z}{\rightarrow}_{c!} y_0} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c?} y_0}{x_0 + x_1 \stackrel{z}{\rightarrow}_{c?} y_0}$$

$$\frac{x_0 \rightarrow_{\tau} y_0}{x_0 \mid\mid x_1 \rightarrow_{\tau} y_0 \mid\mid x_1} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c?} y_0 \quad x_1 \stackrel{z}{\rightarrow}_{c!} y_1}{x_0 \mid\mid x_1 \rightarrow_{\tau} y_0 \mid\mid y_1} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c!} y_0}{x_0 \mid\mid x_1 \stackrel{z}{\rightarrow}_{c!} y_0 \mid\mid x_1} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c?} y_0}{x_0 \mid\mid x_1 \stackrel{z}{\rightarrow}_{c?} y_0 \mid\mid x_1}$$

$$\frac{x_0 \rightarrow_{\tau} y_0}{x_0 \setminus c \rightarrow_{\tau} y_0 \setminus c} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c'!} y_0}{x_0 \setminus c \stackrel{z}{\rightarrow}_{c'!} y_0 \setminus c}\, c \neq c' \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c'?} y_0}{x_0 \setminus c \stackrel{z}{\rightarrow}_{c'?} y_0 \setminus c}\, c \neq c'$$

$$\frac{x_0 \rightarrow_{\tau} y_0}{x_0[S] \rightarrow_{\tau} y_0[S]} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c!} y_0}{x_0[S] \stackrel{z}{\rightarrow}_{S(c)!} y_0[S]} \qquad \frac{x_0 \stackrel{z}{\rightarrow}_{c?} y_0}{x_0[S] \stackrel{z}{\rightarrow}_{S(c)?} y_0[S]}$$

**Figure 9.1** Deduction Rules for CHOCS

send $\stackrel{t}{\rightarrow}_{c!}$ and receive $\stackrel{t}{\rightarrow}_{c?}$ transitions and a nullary internal action $\rightarrow_{\tau}$ transition. Substitution transition $p \stackrel{p'}{\rightarrow}_{/a} p''$ stands for "substituting $a$ with $p'$ in $p$ results in $p''$". Send transition $p \stackrel{p'}{\rightarrow}_{c!} p''$ means that process $p$ emits process $p'$ along channel $c$ and arrives in $p''$, similarly $p \stackrel{p'}{\rightarrow}_{c?} p''$ means that $p$ receives $p'$ along channel $c$ and becomes $p''$. No predicates are used in the TSS of CHOCS.

Deduction rules of the CHOCS semantics are given in Figure 9.1. For brevity, we have omitted the rules dedicated to commutativity of choice and parallel composition. Also, we assume that processes are written in such a way that the substitution happening in the receive rule avoids capture of bound atoms. This can be dealt with explicitly in our SOS framework (cf. [21]) but it will only clutter our presentation and hence we dispense with it.

We also recall stratification from Definition 3.9. We assume all TSS's under study are stratified and consequently, induce a unique stable model.

### 9.3.1 Bisimilarity

Due to the slight change in the notation we use in this chapter (by the introduction of terms as labels), we re-visit the notion of strong bisimilarity. In the following definitions, we write $\mathcal{L}$ for the set of finite lists of terms.

**Definition 9.2 (Strong Bisimilarity [102])** Given a TSS $(\Sigma, V, Rel, Pr, D)$ which induces a unique set of transition relations and predicates, a relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is a *strong simulation* relation if and only if $\forall_{p,q \in \mathcal{C}}\ pRq \Rightarrow$

1. $\forall_{r \in Rel, L \in \mathcal{L}, p' \in \mathcal{C}}\ p \xrightarrow{L}_r p' \Rightarrow \exists_{q' \in \mathcal{C}}\ q \xrightarrow{L}_r q' \wedge p'Rq'$;
2. $\forall_{P \in Pr, L \in \mathcal{L}}\ P(L)p \Rightarrow P(L)q$.

A *strong bisimulation* relation is a symmetric strong simulation relation. Closed terms $p$ and $q$ are strongly bisimilar, denoted by $p \underline{\leftrightarrow}_s q$, if and only if there exists a strong bisimulation relation $R$ such that $pRq$.

We treat this notion in Section 9.4 and there, we formulate a congruence meta-theorem for it in Theorem 9.12.

On one hand, our SOS framework allows for processes as labels. On the other hand processes are usually considered important up to their behavior (and not up to their syntax). Hence, it seems more natural to use a different notion of bisimilarity, rather than the strong one, which not only relates the behavior of source and target processes but also the behavior of label processes. This way, we come to the notion of higher order bisimilarity defined below.

**Definition 9.3 (Higher Order Bisimilarity)** Given a TSS $(\Sigma, V, Rel, Pr, D)$ which induces a unique set of transition relations and predicates, a relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is a *higher order simulation* relation if and only if $\forall_{p,q \in \mathcal{C}}\ pRq \Rightarrow$

1. $\forall_{r \in Rel, L \in \mathcal{L}, p' \in \mathcal{C}}\ p \xrightarrow{L}_r p' \Rightarrow \exists_{L' \in \mathcal{L}, q' \in \mathcal{C}}\ \ q \xrightarrow{L'}_r q' \wedge LRL' \wedge p'Rq'$;
2. $\forall_{P \in Pr, L \in \mathcal{L}}\ P(L)p \Rightarrow \exists_{L' \in \mathcal{L}}\ \ P(L')q \wedge LRL'$.

A *higher order bisimulation* relation is a symmetric higher order simulation relation. Closed terms $p$ and $q$ are higher order bisimilar, denoted by $p \underline{\leftrightarrow}_h q$, if and only if there exists a higher order bisimulation relation $R$ such that $pRq$.

We treat this notion in Section 9.5 and the corresponding congruence results are given in Theorem 9.20.

Note that higher order bisimilarity is usually required to be closed under substitution of atoms. Here, we do not add this requirement for the sake of generality but in the coming examples, we show that this additional constraint can easily be coded in the semantic model.

It is also worth noting that higher order bisimilarity, though more natural in our setting, does not make strong bisimilarity obsolete. In some cases, the labels have a syntactic structure and use terms from the language but do not show any behavior, or alternatively, scrutinizing their behavior is a very complex task. In other words, not always terms on the labels are processes or treated as such. In cases, where labels are indeed terms but do not show any observable behavior, all labels are considered equal from a bisimilarity viewpoint and hence higher order bisimilarity renders very weak and impractical. Thus, presenting a meta-theorem for congruence of bisimilarity is interesting even in the presence of terms as labels.

As one might expect, higher order bisimilarity is strictly coarser than strong bisimilarity, i.e., it identifies more processes and examples of this are shown in the remainder. In Section 9.5, we also give some sufficient criteria for the two notions to coincide.

## 9.3.2 Congruence for Bisimilarity

None of the above mentioned notions of bisimilarity are necessarily a congruence. In the rest of this chapter, we endeavor to find sufficient conditions that guarantee them to be a congruence. After all, it turns out that the sufficient conditions for the two notions are somewhat different. A natural question is whether this difference is genuine or not. In the following two examples we show that the notions of congruence for these two equivalences are indeed unrelated, i.e., for neither of the two equivalences, congruence for one implies congruence for the other.

**Example 9.4** $\dfrac{}{f(a) \xrightarrow{a}_r a} \quad \dfrac{}{a \xrightarrow{a}_r a} \quad \dfrac{}{b \xrightarrow{b}_r b}$

Consider the above set of deduction rules defined on the signature $a$, $b$ and $f(\_)$. In the above TSS, it holds that $a \underline{\leftrightarrow}_h b$ but not $f(a) \underline{\leftrightarrow}_h f(b)$ since $f(a)$ can make an $r$-transition with label $a$ but $f(b)$ cannot make any transition. Higher order bisimilarity is not a congruence for the above TSS. As for strong bisimilarity, it does not hold that $a \underline{\leftrightarrow}_s b$ in the first place and hence, strong bisimilarity is trivially a congruence.

**Example 9.5** $\dfrac{}{f(a) \xrightarrow{a}_r a} \quad \dfrac{}{f(b) \xrightarrow{b}_r a} \quad \dfrac{}{a \xrightarrow{a}_r a} \quad \dfrac{}{b \xrightarrow{a}_r b}$

Consider the above set of deduction rules defined on the same signature as of Example 9.4. This time, higher order bisimilarity is a congruence since $a \underline{\leftrightarrow}_h b$ and $f(a) \underline{\leftrightarrow}_h f(b)$. However, strong bisimilarity is not a congruence since $a \underline{\leftrightarrow}_s b$ but not $f(a) \underline{\leftrightarrow}_s f(b)$.

## 9.4 Congruence for Strong Bisimilarity

In this section, we propose a syntactic restriction on TSSs, in the form of a format, that guarantees strong bisimilarity is a congruence. To begin with, we define the auxiliary notion of volatile operators.

### 9.4.1 Volatile Operators

Due to the possible interaction between terms and labels, for some operators, it is essential to make sure that transitions with these operators (as labels) are always possible under the change of their arguments by bisimilar ones. First, we give a simple example motivating this concept and then we present the formal definition.

**Example 9.6** $\dfrac{a \xrightarrow{g(x)}_r y}{f(x) \xrightarrow{a}_{r'} y} \quad \dfrac{}{a \xrightarrow{g(a)}_r a} \quad \dfrac{}{b \xrightarrow{g(a)}_r a}$

Consider the above TSS with $a$ and $b$ as constants and $f$ and $g$ as unary function symbols. It holds that $a \mathrel{\underline{\leftrightarrow}}_s b$ but it does not hold that $f(a) \mathrel{\underline{\leftrightarrow}}_s f(b)$ and hence strong bisimilarity is not a congruence.

In this case, we call $g$ *volatile* for $r$ transitions because in the premise of the leftmost rule, $g$ appears as a label with an argument that comes from the source of the conclusion of this rule and as such can be replaced by different terms. In order for strong bisimilarity to be a congruence, we require that $r$-transitions with $g$ in the label should be indifferent to replacing arguments of $g$ by bisimilar ones. However, this is clearly not the case for the middle and rightmost rules since for both an $r$ transition with $g(a)$ is allowed while the same transitions with $g(b)$ are prohibited, thus causing the anomaly.

**Definition 9.7 (Volatile Operators)** Given a TSS $(\Sigma, V, Rel, Pr, D)$ an operator $f \in \Sigma$ is called *volatile* for $r \in Rel$ (similarly for $P \in Pr$) when there exists a rule $d \in D$ of the following form:

$$\frac{\{P_i(L_i)t_i \;\text{ or }\; t_i \xrightarrow{L_i}_{r_i} t'_i \mid i \in I\} \quad \{\neg P_j(L_j)t_j \;\text{ or }\; t_j \stackrel{L_j}{\nrightarrow}_{r_j} \mid j \in J\}}{P'(L)t \;\text{ or }\; t \xrightarrow{L}_{r'} t'}$$

and $f(\overrightarrow{t}_{ar(f)-1})$ is a subterm of a component of $L_m$ for some $m \in I \cup J$ such that $r = r_m$ $(P = P_m)$ and $vars(\overrightarrow{t}_{ar(f)-1}) \cap vars(t) \neq \emptyset$ or $\exists_{i \in I} vars(\overrightarrow{t}_{ar(f)-1}) \cap vars(t'_i) \neq \emptyset$.

Informally speaking, in the above definition, we call operator $f$ volatile if in some deduction rule it appears in the label of a premise in such a way that it has a parameter from the source of the conclusion or from a target of a premise. It follows trivially from the above definition that no constant is volatile.

### 9.4.2 The Promoted PANTH Format

Next, we formulate our congruence format for strong bisimilarity.

**Definition 9.8 (The Promoted PANTH Format)** A deduction rule is in the promoted PANTH format when it is of the following form

$$\frac{\{P_i(L_i)t_i \;\text{ or }\; t_i \xrightarrow{L_i}_{r_i} y_i \mid i \in I\} \quad \{\neg P_j(L_j)t_j \;\text{ or }\; t_j \overset{L_j}{\nrightarrow}_{r_j} \mid j \in J\}}{P(L)f(\overrightarrow{x}_{ar(f)-1}) \;\text{ or }\; f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{L}_r t'}$$

and all the variables $x_i$ and $y_j$ ($0 \leq i < ar(f)$ and $j \in I$) and the variables in $L$ are pairwise distinct, if a component of $L_k$ ($k \in I \cup J$) is a variable (i.e., does not have any function symbol) then it is not among $x_i$'s and $y_j$'s and for all components $t''$ of $L$

1. if $t''$ contains a volatile $g \in \Sigma$ for $r$ (for $P$) then $t''$ is of the form $g(\overrightarrow{z}_{ar(g)-1})$ where all $z_i$'s are distinct variables and for all $k \in I \cup J$,
2. if there is a volatile operator for $r$ (for $P$) in the signature and if $t''$ is a variable $z$ then all components of $L_k$ containing $z$ are either $z$ itself or are of the form $g'(\overrightarrow{t}_{ar(g')-1})$ where $g'$ is volatile for $r_k$ (for $P_k$),
3. if there is a volatile operator for $r$ (for $P$) in the signature, for all $i \in I \cup J$, if a component of $L_i$ contains a variable among $x_i$'s, $y_j$'s or the variables of $L$, then $t_i$ contains at least one function symbol (i.e., $t_i$ is not a variable).

A TSS is in the promoted PANTH format when all its deduction rules are.

Observe that if there is no volatile operator in the signature then none of the two checks on the labels are needed. Volatile operators are very rare in process-algebraic formalisms as it can be observed in the coming examples. Hence, most of the times, the above format can be simplified and checks on the labels can be saved. Surprisingly, the promoted tyft/tyxt format is formulated in such a way that all operators can be considered volatile and thus, it turns out to be more restrictive and less expressive than ours. Examples of these phenomena are pointed out next.

**Example 9.9** (Congruence of Strong Bisimilarity for CHOCS) Consider the TSS of CHOCS given in Example 9.1. No operator in this language is volatile. All the deduction rules of this TSS are in the promoted PANTH format but the one concerning the send operator $c!\_.\_$. This rule violates the format by exploiting variable $x_0$ in both the source and the label of the conclusion. All the other rules, having a premise are *not* in the promoted tyft/tyxt format, however, since they have variables as labels of premises. Note that this restriction of promoted tyft/tyxt can be seen as a disadvantage since using this format, one cannot deal with ordinary process algebraic operators (e.g., choice and parallel composition) by replacing variables for constant labels. This restriction is not present in the promoted PANTH format.

Hitherto, one can imagine two scenarios. Either our format is too weak to capture the congruence of strong bisimilarity for CHOCS (since syntactic formats only give sufficient and not necessary conditions) or strong bisimilarity for CHOCS is not a congruence in the first place. Fortunately, the latter is the case and this can be shown by a very simple example.

Consider two processes 0 and $0+0$. It clearly holds that $0 \underline{\leftrightarrow}_s 0+0$ and $0 \underline{\leftrightarrow}_s 0$ but it does not hold that $c!0.0$ is bisimilar to $c!(0+0).0$ as the former can only perform a $\stackrel{0}{\rightarrow}_{c!}$ transition but the latter can only make a $\stackrel{0+0}{\rightarrow}_{c!}$ a transition and 0 and $0+0$ are not (syntactically) the same terms.

However, one can change the language a bit so that strong bisimilarity becomes a congruence. One such approach is presented in [21] and with a proof of more than a page, it is shown that strong bisimilarity in the new language coincides with a notion of higher order bisimilarity [119] in the original semantics and hence, it is concluded that this notion of higher order bisimilarity for the original language is a congruence. In Section 9.5, we propose a congruence format for higher order bisimilarity and using that we give a direct proof for congruence of higher order bisimilarity. So, we do not take the approach of [21] in this section.

Alternatively, in order to make the strong bisimilarity a congruence, we propose to change the send operator as follows. First, we change the syntax of a send operator to be a class of unary send operators $c!p.\_$ for $p \in P$ where $P$ is a fixed set of closed terms. Then, we change the semantics of the send operator and replace it with this rule: $\dfrac{}{c!p.x_0 \stackrel{p}{\rightarrow}_{c!} x_0}$.

Note that in the above rule the $p$ in the source of the conclusion is part of the function symbol while the $p$ in the label is a term. To check that this rule fits in the promoted PANTH format one has to check the following two conditions: first, the set of variables appearing in $p$ and $c!p.x_0$ should be disjoint which holds trivially since the former $p$ is a closed term and second, either $p$ contains no volatile operator or it is of the form $g(\overrightarrow{x}_{ar(g)-1})$ for a volatile $g$. Since the language contains no volatile operator the second obligation is also discharged and hence, we can conclude that strong bisimilarity is a congruence for this slightly modified language. Note that one cannot get a similar result by using the promoted tyft/tyxt format for it only allows for labels of the form $x$ or $g(\overrightarrow{x}_{ar(g)-1})$ in the conclusion.

Next, by a simple and abstract example, we show that our format is strictly more expressive than the promoted tyft/tyxt format of [21].

**Example 9.10** $\dfrac{x \stackrel{z}{\rightarrow}_r y}{f(x) \stackrel{z}{\rightarrow}_r y} \quad \dfrac{}{a \stackrel{f(a)}{\rightarrow}_r b} \quad \dfrac{}{b \stackrel{f(a)}{\rightarrow}_r b}$

Consider a TSS defined by signature $\{a, b, f(\_)\}$, a transition relation $\rightarrow_r$, no predicate and the deduction rules given above. None of the three deduction rules are in the promoted tyft/tyxt format while they are all in the promoted PANTH format and one can check that strong bisimilarity is indeed a congruence. Our claim is that there exists no TSS in the promoted tyft/tyxt format that induces the same transition relation as the one induced by the above TSS.

The proof of our claim is quite simple and follows from the proof of Theorem 3 in [21]. There, it is shown that, for a TSS in the promoted tyft/tyxt format, for all terms $f(\overrightarrow{p}_{ar(f)-1})$ and $g(\overrightarrow{q}_{ar(g)-1})$ if there exists $p' \in \mathcal{C}$ and $\overrightarrow{p'}_{ar(f)-1}, \overrightarrow{q'}_{ar(g)-1} \in \mathcal{L}$ such that $f(\overrightarrow{p}_{ar(f)-1}) \stackrel{g(\overrightarrow{q}_{ar(g)-1})}{\rightarrow_r} p'$, $\overrightarrow{p}_{ar(f)-1} \underline{\leftrightarrow}_s \overrightarrow{p'}_{ar(f)-1}$ and $\overrightarrow{q}_{ar(g)-1} \underline{\leftrightarrow}_s \overrightarrow{q'}_{ar(g)-1}$ then there exists a $p'' \in \mathcal{C}$ such that $f(\overrightarrow{p'}_{ar(f)-1}) \stackrel{g(\overrightarrow{q'}_{ar(g)-1})}{\rightarrow_r} p''$. Getting back to our example, suppose that there exists a TSS in the promoted tyft/tyxt format that induces the same transition relation as the one induced by the above TSS. Then, since $a \underline{\leftrightarrow}_s b$ and $f(a) \stackrel{f(a)}{\rightarrow_r} b$, it should hold that $f(b) \stackrel{f(b)}{\rightarrow_r} p''$ for some $p'' \in \mathcal{C}$ such that $b \underline{\leftrightarrow}_s p''$. But note that in the transition relation induced by the above TSS, no transition with label $f(b)$ is provable.

### 9.4.3 Characteristic Theorem

Common to [21], we impose an extra constraint on the promoted PANTH format to prove congruence, namely the well-foundedness of the TSS under consideration.

**Definition 9.11 (P-Well-Foundedness)** For a deduction rule, the *p-variable ordering* $\leq_p$ is an ordering among variables from the deduction rule. We write $x \leq_p y$, for two variable $x$ and $y$, when $x$ appears in the source or the label of a premise of the deduction rule and $y$ in the target of the same premise. A TSS is called *p-well-founded* when for all deduction rules in TSS, there is no infinite backward chain of variables with respect to $\leq_p$.

Note that in [46] it has been shown by that the well-foundedness assumption, although being very convenient for our congruence proofs, is not essential for the PANTH format. Indeed, for each non-well-founded TSS in the PANTH format, one can construct a well-founded one in a subset of this format (called NTree rules format) that induces the same transition relations and predicates. We leave it open whether the result of [46] carries over to our settings or not.

**Theorem 9.12** (Congruence for Promoted PANTH) For a p-well-founded TSS in the promoted PANTH format, strong bisimilarity is a congruence.

*Proof.*

**Proof Outline.** The proof is inspired by the proof of the similar theorem (Theorem 3) in [21]. We take the bisimilarity induced by a TSS in promoted PANTH format and show that its closure under congruence is still a bisimulation relation. From that we conclude that bisimilarity, being the greatest bisimulation relation, is a congruence.

Next, we present the proof outlined above in full detail. Henceforth, without making it explicit, we neglect the presence of predicates. They cause no technical complication in our proofs but the presentation will be uncluttered by neglecting them.

In this chapter, we assumed that all TSS's are stratified and hence uniquely define a set of transition relations and predicates. In [27], it is shown that defining a unique stable model is not sufficient for a TSS in the ntyft/ntyxt format and stratification is an essential condition by itself. We expect the same result to hold for our case and our proofs essentially depend on the concept of stratification. Hence, we formally define it at this point.

**Definition 9.13 (Stratification)** A *stratification* of a transition system specification $tss$ is a function $\mathcal{S}$ from closed positive formulae to an ordinal such that for all deduction rules of $tss$ of the following form:

$$\textbf{(d)}\frac{\{t_i \xrightarrow{L_i}_{r_i} t'_i \mid i \in I\} \quad \{t_j \overset{L'_j}{\nrightarrow}_{r_j} \mid j \in J\}}{t \xrightarrow{L}_{r} t'}$$

and for all closed substitutions $\sigma$, $\forall_{i\in I}\mathcal{S}(\sigma(t_i \xrightarrow{L_i}_{r_i} t'_i)) \leq \mathcal{S}(\sigma(t \xrightarrow{L}_{r} t'))$ and $\forall_{j\in J, t'_j\in \mathcal{T}}$ $\mathcal{S}(\sigma(t_j \xrightarrow{L_j}_{r_j} t'_j)) < \mathcal{S}(\sigma(t \xrightarrow{L}_{r} t'))$. A transition system specification is called *stratified* if and only if there exists a stratification function for it.

It has been shown in [61] that if a TSS is stratified then it has a *target-independent stratification*, i.e., a stratification that yields the same ordinal for all possible targets. Henceforth, for the TSS's under consideration, we assume and use stratification functions $\mathcal{S}(p, r, L)$ that only take a source (closed term) $p$, a transition relation $r$ and a label (list of closed terms) $L$ as arguments.

Suppose that $tss = (\Sigma, V, Rel, Pr, D)$ in the promoted PANTH format is stratified and thus has a unique stable model. Also, let $\underline{\leftrightarrow}_s$ indicate the strong bisimilarity relation induced by $tss$ and $\tilde{R}$ be the smallest relation satisfying the following constraints:

1. $\underline{\leftrightarrow}_s \subseteq \tilde{R}$;
2. $\forall_{f\in\Sigma}\forall_{\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1}\in\mathcal{C}}\ \overrightarrow{p}_{ar(f)-1}\ \tilde{R}\ \overrightarrow{q}_{ar(f)-1}$
   $\Rightarrow f(\overrightarrow{p}_{ar(f)-1})\ \tilde{R}\ f(\overrightarrow{q}_{ar(f)-1})$.

It is easy to check that $\tilde{R}$ is reflexive and commutative (since $\underline{\leftrightarrow}_s$ is).

If we prove that $\tilde{R}$ is a bisimulation relation then we can conclude that $\underline{\leftrightarrow}_s$ is a congruence since $\underline{\leftrightarrow}_s \subseteq \tilde{R}$ and $\underline{\leftrightarrow}_s$ is the greatest bisimulation relation (thus, $\tilde{R} \subseteq \underline{\leftrightarrow}_s$) and hence, $\underline{\leftrightarrow}_s = \tilde{R}$.

Instead of proving that $\tilde{R}$ is a strong bisimulation relation, we prove the following stronger claim.

**Claim.** For arbitrary $f \in \Sigma$, $p$, $q$, $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1} \in \mathcal{C}$,

1. If $p\tilde{R}q$, $\forall_{p'' \in \mathcal{C}, r \in Rel, \overrightarrow{p'}_n \in \mathcal{L}}$, such that $r$ is of some arity $n$, $p \xrightarrow{\overrightarrow{p'}_n}_r p'' \Rightarrow \exists_{q'' \in \mathcal{C}}$ $q \xrightarrow{\overrightarrow{p'}_n}_r q'' \wedge p''\tilde{R}q''$;
2. and furthermore, if $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(\overrightarrow{q}_{ar(f)-1})$ for an arbitrary function symbol $f$ and $\overrightarrow{p}_{\overrightarrow{f}-1}\tilde{R}\overrightarrow{q}_{\overrightarrow{f}-1}$, $\forall_{p' \in \mathcal{C}, r \in Rel, \overrightarrow{p'}_n \in \mathcal{L}}$, such that $r$ is of some arity $n$, $p \xrightarrow{\overrightarrow{p'}_n}_r p'' \Rightarrow \forall_{\overrightarrow{q'}_n \in \mathcal{C}}$ such that for each component $q'_i$ of $\overrightarrow{q'}_n$, either $p'_i = q'_i$ or $p'_i = g(\overrightarrow{p''}_{ar(g)-1})$, $q'_i = g(\overrightarrow{q''}_{ar(g)-1})$, $g$ is a volatile operator for $r$ and $\overrightarrow{p''}_{ar(g)-1}\tilde{R}\overrightarrow{q''}_{ar(g)-1}$, it holds that $\exists_{q'' \in \mathcal{C}}$ $q \xrightarrow{\overrightarrow{q'}_n}_r q''$ and $p''\tilde{R}q''$.

and two other symmetric conditions for the transition of $q$, the proof of which we omit due to the symmetric structure of $\tilde{R}$.

Note that if we prove the above claim then the transfer conditions for strong bisimilarity follow vacuously from the first item (and its symmetric counterpart).

We prove the claim (both of the above items in parallel) by a transfinite induction on the measure $S(p, q, r, \overrightarrow{p'}_n, \overrightarrow{q'}_n) = \mathcal{S}(p, r, \overrightarrow{p'}_n) + \mathcal{S}(q, r, \overrightarrow{q'}_n)$, i.e., we assume that for all instances transfer conditions for $p$ and $q$ having a measure less than $\beta$ the claim holds, we take a condition with measure $\beta$ and prove that it indeed holds.

Without loss of generality, we assume that $\overrightarrow{l}_n$ and $\overrightarrow{l'}_n$ have only one component (i.e., $n = 1$; the case for nullary relations is simpler while the case for $n$-ary relations is equally difficult but requires a more complicated presentation). Hence, we assume that transitions of $p$ and $q$ are of the form $p \xrightarrow{p''}_r p'$ and $q \xrightarrow{q''}_r q'$.

Inside the transfinite induction we use an induction on the depth of the proof for the transition of $p$. (The base cases of this induction is a special case of the induction step and hence, we dispense with re-stating it).

We proceed with a case distinction based on the structure of $\tilde{R}$.

- If $p\tilde{R}q$ is due to $p \underline{\leftrightarrow}_s q$ then we only concentrate on the first item of the claim and the second item will be covered by the proof in the following case. But the proof of the first item is obvious since it follows immediately from $p \underline{\leftrightarrow}_s q$ (Definition 9.2) that $q \xrightarrow{p''}_r q'$ for some $q'$ such that $p' \underline{\leftrightarrow}_s q'$ and thus $p'\tilde{R}q'$.

- If $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(\overrightarrow{q}_{ar(f)-1})$ and $\overrightarrow{p}_{ar(f)-1}\tilde{R}\overrightarrow{q}_{ar(f)-1}$, then we focus on the proof of the second item which covers the first item if the labels are taken to be equal.

  The last deduction rule applied in the proof tree is due to a closed substitution $\sigma$ and an $f$-defining rule (i.e., with $f$ in the source of the conclusion) **(d)** of the following form (N.B. If the rule has a variable as the source of the conclusion, a simpler line of reasoning leads to the same conclusions and hence we dispense with repeating the arguments):

$$\textbf{(d)}\frac{\{t_i \xrightarrow{t'_i}_{r_i} y_i \mid i \in I\} \quad \{t_j \overset{t'_j}{\nrightarrow}_{r_j} \mid j \in J\}}{t \xrightarrow{t''}_r t'}$$

  where $t$ is of the form $f(\overrightarrow{x}_{ar(f)-1})$ and $\sigma(\overrightarrow{x}_{ar(f)-1}) = \overrightarrow{p}_{ar(f)-1}$, $\sigma(t'') = p''$ and $\sigma(t') = p'$. We aim at defining a closed substitution $\sigma'$ such that $\sigma$ and $\sigma'$ respect $\tilde{R}$, so that we can prove the desired transition for $q$. To start with we define $\sigma'_0$ as the basis for $\sigma'$, and for that we distinguish the following three cases:

  1. If $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some non-volatile operator $g$ , then define:
  $$\sigma'_0(x) = \begin{cases} q_i & x = x_i \\ \sigma(x) & x \notin \{x_i, y_j | 0 \leq i < ar(f), j \in I\} \end{cases}$$
  2. If $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some volatile operator $g$ and $t''$ is a variable $z$, then we have to prove the transition of $q$ for an arbitrary $q'' = g(\overrightarrow{q''}_{ar(g)-1})$ such that $\overrightarrow{p''}_{ar(g)-1}\tilde{R}\overrightarrow{q''}_{ar(g)-1}$, then take:
  $$\sigma'_0(x) = \begin{cases} q_i & x = x_i \\ q'' & x = z \\ \sigma(x) & x \notin \{x_i, z, y_j | 0 \leq i < ar(f), j \in I\} \end{cases}$$
  3. If $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some volatile operator $g$ and $t''$ is a term $g(\overrightarrow{z}_{ar(g)-1})$, then we have to prove the transition of $q$ for an arbitrary $q'' = g(\overrightarrow{q''}_{ar(g)-1})$ such that $\overrightarrow{p''}_{ar(g)-1}\tilde{R}\overrightarrow{q''}_{ar(g)-1}$, then take:
  $$\sigma'_0(x) = \begin{cases} q_i & x = x_i \\ q''_i & x = z_i \\ \sigma(x) & x \notin \{x_i, z_j, y_k | 0 \leq i < ar(f), 0 \leq j < ar(g), k \in I\} \end{cases}$$

Note that in all of the above cases $\sigma$ and $\sigma'_0$ respect $\tilde{R}$ on their common domain. Now, we aim at completing the definition of $\sigma'$ by defining it on the set $Y \doteq \{y_i \mid i \in I\}$ in such a way that $\sigma(y)\tilde{R}\sigma'(y)$ for all $y \in Y$. We do so in a step by step fashion, resulting in a new $\sigma'_i$ at each step, while preserving the aforementioned constraint. To do this, we take a premise $t_j \xrightarrow{t'_j}_{r_j} y_j$ of which the variables in the source and label are all defined in $\sigma_i$. Note that such a premise should exist initially and at each step due to the p-well-foundedness assumption. We give the following general construction for arriving at a $\sigma'_{i+1}$.

We distinguish two cases: either $t'_j$ does not contain a variable among $x_i$'s, $y_j$'s and variables of $t''$ (for some $i \in I$ and $k \in J$) or it does.

If $t'_j$ does not contain a variable among $x_i$'s, $y_j$'s and variables of $t''$, then it holds that $\sigma(t_j)\tilde{R}\sigma'_i(t_j)$ and $\sigma(t'_j) = \sigma'_i(t'_j)$ and from the induction hypothesis on the depth of the proof (the first item of the claim), it follows that there exists a $q'_j$ such that $\sigma'_i(t_j) \xrightarrow{\sigma'_i(t'_j)}_{r_j} q'_j$ and $\sigma(y_j)\tilde{R}q'_j$. We define $\sigma'_{i+1} = \sigma'_i[y_j \mapsto q'_j]$ and $\sigma$ and $\sigma'_{i+1}$ respect $\tilde{R}$ on their common domain.

If it does, then it follows from item 3 in Definition 9.8, $t_j$ is contains a function symbol $k$ and is of the form $k(\overrightarrow{s}_{ar(k)-1})$. Also, it follows from item 2 of the same definition that either $t'_j = t'' = z$ or $t'_j = g(\overrightarrow{z}_(ar(g)-1)$ for some volatile operator $g$. In both cases, it follows from the construction of $\sigma'_i$ that $\sigma(\overrightarrow{s}_{ar(k)-1}\ \tilde{R}\ \sigma'_i(\overrightarrow{s}_{ar(k)-1}$ and $\sigma(\overrightarrow{z}_{ar(g)-1})\ \tilde{R}\ \sigma'_i(\overrightarrow{z}_{ar(g)-1})$. Since $\sigma(t_j) \xrightarrow{\sigma(t'_j)}_{r_j} \sigma(y_j)$ has a proof of depth $n-1$ and the sum of stratification measures does not increase from the conclusion to the two premises, the hypothesis of the induction on the proof depth (the second item of the claim) applies and it follows that $\sigma'_i(t_j) \xrightarrow{\sigma'_i(t'_j)}_{r_j} q'_j$ for some $q'_j \in \mathcal{C}$ and $\sigma(y_j)\tilde{R}q'_j$. In this case, we define $\sigma'_{i+1} = \sigma'_i[y_j \mapsto q'_j]$ and $\sigma$ and $\sigma'_{i+1}$ respect $\tilde{R}$ on their common domain.

Substitution $\sigma'$ is defined as the union of all $\sigma'_i$'s. Since the procedure is monotonic on the domain of $\sigma'_i$'s w.r.t. the set inclusion ordering, it follows from Tarski's fixpoint theorem that such a $\sigma'$ indeed exists.

Using $\sigma'$, we have a proof for all positive premises of **(d)**. Also, negative premises are satisfied by the stable model of $tss$, since otherwise, there would be a transition $\sigma'(t'_j) \xrightarrow{\sigma'(t'_j)}_{r_j} q'_j$ provable for some $q'_j$. On one hand, if $t'_j$ contains a variable from $x_i$'s, $y_i$'s or variables of $t''$, then $\sigma'(t'_j)$ should be of the form $g_j(\overrightarrow{p''}_{ar(g_j)-1})$ for a volatile operator $g_j$ for $r_j$ (and $t_j$ contains a function symbol) and otherwise $\sigma(t''_j) = \sigma'(t''_j)$. On the other hand, $\mathcal{S}(\sigma'(t'_j), r_j, \sigma'(t''_j)) < \mathcal{S}(\sigma'(t), r, \sigma'(t''))$, $\mathcal{S}(\sigma(t'_j), r_j, \sigma(t''_j)) < \mathcal{S}(\sigma(t), r, \sigma(t''))$ and hence $\mathcal{S}(\sigma(t'_j), r_j, \sigma(t''_j)) + \mathcal{S}(\sigma'(t'_j), r_j, \sigma'(t''_j)) < \beta$. Hence, the induction hypothesis applies and there should be a transition $\sigma(t_j) \xrightarrow{\sigma(t''_j)}_r p'_j$ provable for some $p'_j$ contradicting the provability of the transition for $p$.

In conclusion, using $\sigma'$ and deduction rule **(d)**, we can derive a transition $\sigma'(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{t''}_r t')$ or $f(\overrightarrow{q'}_{ar(f)-1}) \xrightarrow{\sigma'(t'')}_r \sigma'(t')$ where $\sigma'(t'')$ is either $p''$ or $g(\overrightarrow{q''}_{ar(f)-1})$ depending on the structure of $p''$ and since $\sigma'$ respects $\tilde{R}$ by construction, it holds that $\sigma(t'')\tilde{R}\sigma'(t'')$ or $p'\tilde{R}\sigma'(t'')$.

$\boxtimes$

In order to generalize the result of [21] in the setting without negative premises (and with all operators considered volatile), we need to get rid of item 3 in Definition 9.8 (of the promoted PANTH format). The following theorem realizes this goal.

**Theorem 9.14** If a positive and p-well-founded TSS satisfies all the constraints of Definition 9.8 but item 3, then bisimilarity is a congruence.

*Proof.*

**Proof Outline.** The proof goes along the same line as the proof of Theorem 9.12 with two main differences. First, the induction is only on the proof depth of each of the transitions instead of the sum of stratification measures and second, the congruence closure $\tilde{R}$ now contains a phrase which resembles the transitivity property. The detail of the proof is given below.

Let $\underline{\leftrightarrow}_s$ indicate the strong bisimilarity relation induced by the TSS under consideration and $\tilde{R}$ be the smallest relation satisfying the following constraints:

1. $\underline{\leftrightarrow}_s \subseteq \tilde{R}$;
2. $\forall_{p,p_1,q \in \mathcal{C}}\ p\tilde{R}p_1 \wedge p_1 \underline{\leftrightarrow}_s q \Rightarrow p\tilde{R}q$;
3. $\forall_{f \in \Sigma} \forall_{\overrightarrow{p}_{ar(f)-1}, \overrightarrow{q}_{ar(f)-1} \in \mathcal{C}}\ \overrightarrow{p}_{ar(f)-1}\ \tilde{R}\ \overrightarrow{q}_{ar(f)-1} \Rightarrow$
   $f(\overrightarrow{p}_{ar(f)-1})\ \tilde{R}\ f(\overrightarrow{q}_{ar(f)-1})$.

It is easy to check that $\tilde{R}$ is reflexive (since $\underline{\leftrightarrow}_s$ is).

We prove the following claim for $\tilde{R}$.

**Claim.** For arbitrary $f \in \Sigma$, $p$, $q$, $\overrightarrow{p}_{ar(f)-1}$, $\overrightarrow{q}_{ar(f)-1} \in \mathcal{C}$, if $p\tilde{R}q$,

1. $\forall_{p'' \in \mathcal{C}, r \in Rel, \overrightarrow{p'}_n \in \mathcal{L}}$, such that $r$ is of some arity $n$, $p \xrightarrow{\overrightarrow{p'}_n}_r p'' \Rightarrow \forall_{\overrightarrow{q'}_n \in \mathcal{L}}$ such that for each component $q'_i$ of $\overrightarrow{q'}_n$, either $p'_i = q'_i$ or $p'_i = g(\overrightarrow{p''}_{ar(g)-1})$, $q'_i = g(\overrightarrow{q''}_{ar(g)-1})$, $g$ is a volatile operator for $r$ and $\overrightarrow{p''}_{ar(g)-1} \tilde{R} \overrightarrow{q''}_{ar(g)-1}$, it holds that $\exists_{q'' \in \mathcal{C}}\ q \xrightarrow{\overrightarrow{q'}_n}_r q'' \wedge p'' \tilde{R} q''$;
2. $\forall_{q'' \in \mathcal{C}, r \in Rel, \overrightarrow{q'}_n \in \mathcal{L}}$, such that $r$ is of some arity $n$, $q \xrightarrow{\overrightarrow{q'}_n}_r q'' \Rightarrow \forall_{\overrightarrow{p'}_n \in \mathcal{L}}$ such that for each component $p'_i$ of $\overrightarrow{p'}_n$, either $q'_i = p'_i$ or $q'_i = g(\overrightarrow{q''}_{ar(g)-1})$, $p'_i = g(\overrightarrow{p''}_{ar(g)-1})$, $g$ is a volatile operator for $r$ and $\overrightarrow{q''}_{ar(g)-1} \tilde{R} \overrightarrow{p''}_{ar(g)-1}$, it holds that $\exists_{q'' \in \mathcal{C}}\ q \xrightarrow{\overrightarrow{q'}_n}_r q'' \wedge q'' \tilde{R} p''$;

If we prove the above claim, it follows that the symmetric closure of $\tilde{R}$, denoted by $R^*$ also satisfies both of the above items and hence, $R^*$ is a bisimulation relation containing $\underline{\leftrightarrow}_s$ and thus $R^*$ and $\underline{\leftrightarrow}_s$ coincide, hence $\underline{\leftrightarrow}_s$ is a congruence.

For the sake of brevity, we only prove the first item of the above claim and the proof of the second item follows the same structure. Also, we confine ourselves to the setting were the labels contain only a single term. Transitions of $p$ are thus of the form $p \xrightarrow{p''}_r p'$.

We start with an induction on the depth of the proof for the transition of $p$ and proceed with another induction on the structure of $\tilde{R}$.

1. If $p\tilde{R}q$ is due to $p \underline{\leftrightarrow}_s q$ then depending on the outermost symbol in $p''$ the following two cases can be distinguished:
   (a) Either $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some non-volatile operator $g$ for $r$ then it follows immediately from $p \underline{\leftrightarrow}_s q$ (Definition 9.2) that $q \xrightarrow{p''}_r q'$ for some $q'$ such that $p' \underline{\leftrightarrow}_s q'$ and thus $p'\tilde{R}q'$;
   (b) Or, $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some volatile operator $g$ for $r$. This transition of $p$ should be due to a deduction rule **(d)** in the promoted PANTH format of the following form

$$\textbf{(d)}\frac{\{t_i \xrightarrow{t'_i}_{r_i} y_i \mid i \in I\} \quad \{t_j \stackrel{t'_j}{\nrightarrow}_{r_j} \mid j \in J\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{t''}_r t'}$$

   and a substitution $\sigma$ such that $q = \sigma(f(\overrightarrow{x}_{ar(f)-1}))$, $p' = \sigma(t')$ and $g(\overrightarrow{p''}_{ar(g)-1}) = \sigma(t'')$. Since $g$ is a volatile operator for $r$, it follows from the constraints of the promoted PANTH format that either $t''$ is a variable $z$ or it is of the form $g(\overrightarrow{z}_{ar(g)-1})$. To prove the claim, i.e.,

$q \xrightarrow{q''}_r q'$ for $q'' = g(\overrightarrow{q''}_{ar(g)-1})$ and $p'\tilde{R}q'$, we first prove $p \xrightarrow{q''}_r p'_1$ for some $p'_0$ such that $p'\tilde{R}p'_1$ and then using the definition of strong bisimilarity, from $p \underline{\leftrightarrow}_s q$, it follows that $q \xrightarrow{q''}_r q'$ for some $q'$ such that $p'_1 \underline{\leftrightarrow}_s q'$ and by the construction of $\tilde{R}$, we deduce that $p'\tilde{R}q'$.

So, it only remains to prove that $p \xrightarrow{q''}_r p'_1$ for some $p'_1$ such that $p\tilde{R}p'_1$. To this end, we use deduction rule **(d)** and construct a new $\sigma'$ which w.r.t. $\sigma$ respects $\tilde{R}$ and furthermore satisfies $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) = p$, $\sigma'(t'') = q'' = g(\overrightarrow{q''}_{ar(g)-1})$ and $\sigma'(t') = p'$ for some $p'_1$ such that $p'\tilde{R}p'_1$ and all premises with $\sigma'$ applied to them are provable.

If $t''$ is a variable $z$, then we define

$$\sigma'_0(x) = \begin{cases} g(\overrightarrow{q''}_{ar(g)} - 1) & x = z \\ \sigma(x) & x \notin \{z, y_i | i \in I\} \end{cases}$$

and otherwise, if it is of the form $g(\overrightarrow{z}_{ar(f)-1})$ then,

$$\sigma'_0(x) = \begin{cases} q''_i & x = z_i, 0 \leq i < ar(g) \\ \sigma(x) & x \notin vars(t'') \cup \{y_i | i \in I\} \end{cases}$$

We aim at adding variables from $\{y_i \mid i \in I\}$ to the domain of $\sigma'_0$ in a step by step fashion, resulting in a new $\sigma'_i$ at each step, while preserving the constraint $\forall_{x \in dom(\sigma_i)} \sigma(x)\tilde{R}\sigma'_i(x)$. (N.B. hitherto, it holds that $\forall_{x \in dom(\sigma_0)}\ \sigma(x)\ \tilde{R}\ \sigma'_0(x))$ To do this, we take a premise $t_j \xrightarrow{t'_j}_{r_j} y_j$ of which the variables in the source and label are all defined in $\sigma_i$. Note that such a premise should exist initially and at each step due to the p-well-foundedness assumption. We give the following general construction for arriving at a $\sigma'_{i+1}$.

Hence, it follows from the structure if $\sigma'_i$ that $\sigma(t_j)\tilde{R}\sigma'_i(t_j)$ and $\sigma(t'_j)\tilde{R}\sigma'_i(t'_j)$. Furthermore, if $\sigma(t'_j) \neq \sigma'_i(t'_j)$, then it contains a variable amon $z_i$'s or $y_i$'s and hence, it is of the form $g(\overrightarrow{t'}_{ar(g)-1})$ for a volatile operator $g$. Since transition $\sigma(t_j) \xrightarrow{\sigma(t'_j)}_{r_j} \sigma(y_j)$ has a proof of depth $n-1$, the induction hypothesis on the depth of the proof applies and thus, $\sigma'_i(t_j) \xrightarrow{\sigma'_i(t'_j)}_{r_j} p''_j$ for some $p''_j$ such that $\sigma(y_j)\tilde{R}p''_j$. Take $\sigma'_{i+1} = \sigma'_i[y_j \mapsto p''_j]$ and $\sigma$ and $\sigma'_{i+1}$ respect $\tilde{R}$ on their common domain.

Substitution $\sigma'$ is defined as the union of all $\sigma'_i$'s. Since the procedure is monotonic on the domain of $\sigma'_i$'s w.r.t. the set inclusion ordering, it follows from Tarski's fixpoint theorem that such a $\sigma'$ indeed exists.

Using $\sigma'$, we have a proof for all positive premises of **(d)** and hence, using $\sigma'$ and deduction rule **(d)**, we are able to prove the transition

$\sigma'(f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{t''}_r t')$, or $p \xrightarrow{g(\overrightarrow{q'}_{ar(g)-1})}_r \sigma'(t')$ and by the construction of $\sigma'$, it holds that $p'\tilde{R}\sigma'(t')$. As stated before, it follows from the definition of strong bisimilarity and the construction of $\tilde{R}$ that $q \xrightarrow{g(\overrightarrow{q'}_{ar(g)-1})}_r q''$ for some $q''$ such that $p''\tilde{R}q''$.

2. If $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(\overrightarrow{q}_{ar(f)-1})$ and $\overrightarrow{p}_{ar(f)-1}\tilde{R}\overrightarrow{q}_{ar(f)-1}$, then the remainder of the proof is similar to the second case in the first item ($p \underline{\leftrightarrow}_s q$) apart from the first step in defining $\sigma_0$. Next, we give the details of this proof.

   The last deduction rule applied in the proof tree is due to a closed substitution $\sigma$ and an $f$-defining rule (i.e., with $f$ in the source of the conclusion) **(d)** of the following form (N.B. If the rule has a variable as the source of the conclusion, a simpler line of reasoning leads to the same conclusions and hence we dispense with repeating the arguments):

$$\textbf{(d)}\,\frac{\{t_i \xrightarrow{t_i'}_{r_i} y_i \mid i \in I\} \quad \{t_j \overset{t_j'}{\nrightarrow}_{r_j} \mid j \in J\}}{t \xrightarrow{t''}_r t'}$$

   where $t$ is of the form $f(\overrightarrow{x}_{ar(f)-1})$ and $\sigma(\overrightarrow{x}_{ar(f)-1}) = \overrightarrow{p}_{ar(f)-1}$, $\sigma(t'') = p''$ and $\sigma(t') = p'$. We aim at defining a closed substitution $\sigma'$ such that $\sigma$ and $\sigma'$ respect $\tilde{R}$, so that we can prove the desired transition for $q$. To start with we define $\sigma_0'$ as the basis for $\sigma'$, and for that we distinguish the following three cases:

   (a) If $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some non-volatile operator $g$ , then define:

$$\sigma_0'(x) = \begin{cases} q_i & x = x_i \\ \sigma(x) & x \notin \{x_i, y_j | 0 \le i < ar(f), j \in I\} \end{cases}$$

   (b) If $p\tilde{R}q$ is due to the fact that there is a term $p_1 \in \mathcal{C}$ such that $p\tilde{R}p_1$ and $s \underline{\leftrightarrow}_s q$ and $p \xrightarrow{p''}_r p'$, we distinguish the following two cases based on the the form of $p''$.

      i. If $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some non-volatile operator $g$ for $r$ then the hypothesis of the innermost induction (on the structure of $\tilde{R}$) applies and $p_1 \xrightarrow{p''}_r p_1'$ for some $p_1'$ such that $p'\tilde{R}p_1'$. Since $p_1 \underline{\leftrightarrow}_s q$, there exists a $q'$ such that $q \xrightarrow{p''}_r q'$ and $p_1' \underline{\leftrightarrow}_s q'$. It follows from the construction of $\tilde{R}$ that $p'\tilde{R}q'$ and hence, the claim.

      ii. Similarly, if $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some volatile operator $g$ for $r$. Take an arbitrary $\overrightarrow{q''}_{ar(g)-1}$ such that $\overrightarrow{p''}_{ar(g)-1}\tilde{R}\overrightarrow{q''}_{ar(g)-1}$. It again follows from the hypothesis of the innermost induction (on the structure of $\tilde{R}$) that $p_1 \overset{g(\overrightarrow{q''}_{ar(g)-1})}{\longrightarrow_r} p'_1$ for some $p'_1$ such that $p'\tilde{R}p'_1$. Since $s \underline{\leftrightarrow}_s q$, there exists a $q'$ such that $q \overset{g(\overrightarrow{q''}_{ar(g)-1})}{\longrightarrow_r} q'$ and $p'_1 \underline{\leftrightarrow}_s q'$ and by the construction of $\tilde{R}$, $p'\tilde{R}q'$.

(c) If $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some volatile operator $g$ and $t''$ is a variable $z$, then we have to prove the transition of $q$ for an arbitrary $q'' = g(\overrightarrow{q'}'_{ar(g)-1})$ such that $\overrightarrow{p''}_{ar(g)-1}\tilde{R}\overrightarrow{q''}_{ar(g)-1}$, then take:

$$\sigma'_0(x) = \begin{cases} q_i & x = x_i \\ q'' & x = z \\ \sigma(x) & x \notin \{x_i, z, y_j | 0 \leq i < ar(f), j \in I\} \end{cases}$$

(d) If $p'' = g(\overrightarrow{p''}_{ar(g)-1})$ for some volatile operator $g$ and $t''$ is a term $g(\overrightarrow{z}_{ar(g)-1})$, then we have to prove the transition of $q$ for an arbitrary $q'' = g(\overrightarrow{q''}_{ar(g)-1})$ such that $\overrightarrow{p''}_{ar(g)-1}\tilde{R}\overrightarrow{q''}_{ar(g)-1}$, then take:

$$\sigma'_0(x) = \begin{cases} q_i & x = x_i \\ q''_i & x = z_i \\ \sigma(x) & x \notin \{x_i, z_j, y_k | 0 \leq i < ar(f), 0 \leq j < ar(g), k \in I\} \end{cases}$$

Note that in all of the above cases $\sigma$ and $\sigma'_0$ respect $\tilde{R}$ on their common domain. The construction of $\sigma' = \cup\sigma'_i$ remains the same as in the first item and hence, we can derive a transition $\sigma'(f(\overrightarrow{x}_{ar(f)-1}) \overset{t''}{\longrightarrow}_r t')$ or $f(\overrightarrow{q'}_{ar(f)-1}) \overset{\sigma'(t'')}{\longrightarrow_r} \sigma'(t')$ where $\sigma'(t'')$ is either $p''$ or $g(\overrightarrow{q''}_{ar(f)-1})$ depending on the structure of $p''$ and since $\sigma'$ respects $\tilde{R}$ by construction, it holds that $\sigma(t'')\tilde{R}\sigma'(t'')$ or $p'\tilde{R}\sigma'(t'')$.

⊠

## 9.5 Congruence for Higher Order Bisimilarity

### 9.5.1 Persistency

In this section, we seek sufficient syntactic criteria for the higher order bisimilarity induced by a TSS to be a congruence. We begin with an auxiliary definition that has the same spirit as that for volatile operators. It is supposed to capture that the labels of a transition can be replaced by bisimilar ones.

**Definition 9.15 (Persistent Transitions)** Consider a TSS $(\Sigma, V, Rel, Pr, D)$ and a set $Ps$ of tuples $(U, L)$ where $U \in Rel \cup Pr$ and $L \in \mathcal{L}$. We call $Ps$ a *persistent set* when for all $(U, L) \in Ps$ and all deduction rules $d \in D$ if **(d)** has $U$ in its conclusion then it is of the following form:

$$\textbf{(d)}\frac{\{P(L_i)t_i \ \text{ or } \ t_i \xrightarrow{L_i}_{r_i} y_i \mid i \in I\} \quad \{\neg P(L_j)t_j \ \text{ or } \ t_j \overset{L_j}{\nrightarrow}_{r_j} \mid j \in J\}}{U(L')f(\overrightarrow{x}_{ar(f)-1}) \ \text{ or } \ f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{L'}_U t'}$$

where $L = \sigma(L')$ for some substitution $\sigma$ and

1. all $x_i$'s, $y_j$'s ($0 \leq i < ar(f)$ and $j \in I$) and variables appearing in $L'$ are pairwise distinct;
2. for all $k \in I \cup J$, $(r_k, \sigma(L_k)) \in Ps$ (or $(P_k, \sigma(L_k)) \in Ps$).

If a set $Ps$ is persistent and $(U, L) \in Ps$ then we say that *$U$-transitions (predicates) are persistent for $L$ labels*. A transition relation (predicate) is *persistent* if it is persistent for a label of the form $\overrightarrow{z}_n$ where $z_i$'s are distinct variables.

The following theorem gives an idea about the intuition behind persistency.

**Theorem 9.16** If for a TSS all its transition relations and predicates are persistent then:

1. higher order bisimilarity is a congruence;
2. higher order and strong bisimilarity coincide.

We defer the proof of this theorem to the next subsection where we give a proof of congruence for our general rule format. Of course, we do not use this theorem in the proofs of the rule format.

**Example 9.17** (Persistency for CHOCS) Substitution, receive and $\tau$-transitions are all persistent in CHOCS, i.e., substitution and receive are persistent for a variable.

### 9.5.2 Higher Order PANTH Format

Our criteria are formulated as a syntactic format which we call higher order PANTH.

**Definition 9.18 (Higher Order PANTH Format)** A deduction rule is in the higher order PANTH when it is of the following form

$$\frac{\{P(L_i)t_i \ \text{ or } \ t_i \xrightarrow{L_i}_{r_i} y_i \mid i \in I\} \quad \{\neg P(L_j)t_j \ \text{ or } \ t_j \overset{L_j}{\nrightarrow}_{r_j} \mid j \in J\}}{P(L)f(\overrightarrow{x}_{ar(f)-1}) \ \text{ or } \ f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{L}_r t'}$$

where variables $x_i$'s and $y_j$'s ($0 \leq i < ar(f)$ and $j \in I$) are all pairwise distinct and for all $k \in I \cup J$

1. either $r_k$-transitions (predicates) are persistent for $L_k$ labels (Definition 9.15);
2. or otherwise, $k \in I$, $L_k$ is a list of variables $\overrightarrow{z}_m$ which are all distinct among themselves, different from variables in the labels of other non-persistent transitions and predicates and different from $x_i$'s and $y_j$'s.

A TSS is in the higher order PANTH format when all its rules are.

Next, we define the notion of well-foundedness for TSS's in the higher order PANTH format.

**Definition 9.19 (H-Well-Foundedness)** An *h-variable ordering* $\leq_h$ with respect to a deduction rule is an ordering on variables in the deduction rule. For two variables $x$ and $y$, $x \leq_h y$ if $x$ appears in the source of a premise of the rule and $y$ appears in its label or target. A TSS is *h-well-founded* when for all deduction rules in TSS, there is no infinite backward chain of variables with respect to $\leq_h$.

We think that well-foundedness for this format, like for PANTH format, is a convenience for our proofs and is not a necessary ingredient for congruence.

**Theorem 9.20** (Congruence for higher order PANTH) For an h-well-founded TSS in the higher order PANTH format, higher order bisimilarity is a congruence.

*Proof.*

**Proof Outline.** The proof goes along the same lines as the proof of Theorem 9.12. In the proof of theorem 9.12, we tried to build a proof for transitions of terms related by $\tilde{R}$ using the same deduction rule and a newly defined substitution. Here, we follow the same idea, however, the main difference lies in the construction of the substitution. There, the basic substitution evaluated everything but targets of the premises and a procedure was given to make it complete by chasing the chain of premises with respect to the variable ordering. Here, in addition to the targets of all premises, we initially do not evaluate labels of freely-labelled premises. These labels are also to be evaluated while traversing the chain of premises. This difference arises from the fact that in higher order bisimulation, we cannot assume that bisimilar terms make the same transitions with literally the same labels. Thus, here, labels are to be chosen at will by the term making the transition.

A detailed account of the proof is given next.

**Definition 9.21 (Freely Labelled Premises)** For a deduction rule in the higher order PANTH format, positive premises that make use of the second condition of Definition 9.18 are called *freely-labelled premises*. If a positive premise satisfies both of the conditions, it does not matter whether it is considered freely-labelled or not.

Suppose that $tss = (\Sigma, V, Rel, Pr, D)$ is in the higher order PANTH format and is stratified. Hence, it has a unique stable model. Also, let $\underline{\leftrightarrow}_h$ indicate the higher order bisimilarity relation induced by $tss$ and $\tilde{R}$ be the smallest congruence relation containing $\underline{\leftrightarrow}_h$. If we prove that $\tilde{R}$ is a bisimulation relation then we can conclude that $\underline{\leftrightarrow}_h$ is a congruence since $\underline{\leftrightarrow}_h \subseteq \tilde{R}$ and $\underline{\leftrightarrow}_h$ is the greatest higher order bisimulation relation (thus, $\tilde{R} \subseteq \underline{\leftrightarrow}_h$) and hence, $\underline{\leftrightarrow}_h = \tilde{R}$.

To prove that $\tilde{R}$ is a higher order bisimulation relation, we take arbitrary terms $p, q \in \mathcal{C}$ such that $p\tilde{R}q$ and show the following statements $\forall_{r \in Rel, L \in \mathcal{L}}$

1. $\forall_{p' \in \mathcal{C}}$, if $p \xrightarrow{L}_r p' \Rightarrow \exists_{L' \in \mathcal{L}, q' \in \mathcal{C}}\ q \xrightarrow{L'}_r q'$, $L\tilde{R}L' \wedge p'Rq'$;

2. if $r$ is persistent for $L''$, $L = \sigma(L'')$, $p \xrightarrow{L}_r p'$ and $\sigma$ and $\sigma'$ respect $\tilde{R}$ then $\exists_{p'' \in \mathcal{C}}\ q \xrightarrow{\sigma'(L'')}_r p''$, $\wedge\ p'Rp''$;

Note that the last statement is in addition to the transfer conditions for proving bisimilarity but is required for our proof.

We prove the above statements by a transfinite induction on $\mathcal{S}(p, r, L)$. We assume that for all transitions of $p$ with label $L$ such that the above measure is less than some ordinal $\beta$ the above statements hold. Now we take a transitions of $p$ for which the above measure is $\beta$ and prove the transfer conditions.

To simplify matters, we assume that the labels consist of a single term. Hence the transitions of $p$ are of the form $p \xrightarrow{p''}_r p'$.

We proceed with an induction on the depth of the transition of $p$. We dispense with the induction basis as it is a special case of the induction step in which the last deduction rule in the proof tree has no premises.

To prove item 1, we distinguish the following two cases based on the structure of $\tilde{R}$.

If $p\tilde{R}q$ is due to $p \underline{\leftrightarrow}_h q$ then the theorem follows trivially from the definition of higher order bisimilarity, i.e., Definition 9.3.

If $p\tilde{R}q$ is due to the congruence closure of $\underline{\leftrightarrow}_h$ then $p = f(\overrightarrow{p}_{ar(f)-1})$ and $q = f(\overrightarrow{q}_{ar(f)-1})$ and the transition of $p$ should be due to a rule **(d)** in the higher order

PANTH format which has the following form:

$$\frac{\{t_i \xrightarrow{L_i}_{r_i} y_i \mid i \in I\} \quad \{t_j \stackrel{L_j}{\nrightarrow}_{r_j} \mid j \in J\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{L}_r t'}$$

Let $X$ denote the set of variables in the source of the conclusion, $Y$ the set of variables in the target of the premises and $Z$ the set of variables in the labels of freely-labelled premises. Then, we aim at defining a new substitution $\sigma'$ which respects $\tilde{R}$ w.r.t. $\sigma$ and gives us a proof for the transition of $q$. To start with we define $\sigma'_0$ as follows.

$$\sigma'_0(x) = \begin{cases} q_i & x \in X \\ \sigma(x) & x \notin X \cup Y \cup Z \end{cases}$$

Two substitutions $\sigma$ and $\sigma'_0$ respect $\tilde{R}$. It remains to complete the definition of $\sigma'$ by defining it one the variables from $Y \cup Z$.

We continue with a procedure to complete the definition of $\sigma'_0$. The procedure is given in such a way that $\sigma$ and $\sigma'_i$ always respect $\tilde{R}$ on their common domain. Take any $\sigma'_i$ and a premise $t_j \xrightarrow{t'_j}_{r_j} y_j$ of which all the variable in the source are valuated by $\sigma'_i$ and the variable in the target remains to valuated. Either this premise is freely labelled, then $t'_j = z_j$ and it is not valuated by $\sigma'_i$ (due to the acyclicity constraint). Or, $r_j$ is persistent for $t'_j$-labels. In the former case since $\sigma(t_j)\tilde{R}\sigma'_i(t_j)$, it follows from the natural induction hypothesis that there exists $p'_j$ and $p''_j$ such that $\sigma(y_j)\tilde{R}p'_j$ and $\sigma(z_j)\tilde{R}p''_j$ and $\sigma'_i(t_j) \xrightarrow{p''_j}_{r_j} p'_j$, then let $\sigma'_{i+1}$ be $\sigma'_i[z_j \mapsto p''_j][y_j \mapsto y''_j]$ and $\sigma$ and $\sigma'_{i+1}$ respect $\tilde{R}$ on their common domain.

We can now verify whether negative premises do hold with respect to the induced stable model or not. Consider a negative premise $t_j \stackrel{t'_j}{\nrightarrow}_{r_j}$. Suppose that $\sigma'_0(t_j) \stackrel{\sigma'_0(t'_j)}{\nrightarrow}_{r_j}$ does not hold and there exists a $q'_j$ such that $\sigma'_0(t_j) \xrightarrow{\sigma'_0(t'_j)}_{r_j} q'_j$ since $r_j$ transitions are persistent for $t'_j$-labels, it also holds that $\sigma'_0(t_j) \xrightarrow{\sigma(t'_j)}_{r_j} q'_j$. Since $min(\mathcal{S}(\sigma(t_j), r_j, \sigma(t'_j)), \mathcal{S}(\sigma'_0(t_j), r_j, \sigma(t'_j))) \leq \mathcal{S}(\sigma(t_j), r_j, \sigma(t'_j)) < \mathcal{S}(p, r, p'')$, the induction hypothesis applies and hence there should exist a provable transition $\sigma(t_j) \xrightarrow{\sigma(t_j)}_{r_j} p'_j$, which is in contradiction with the provability of the transition for $p$.

Take $\sigma'$ as the union of all $\sigma'_i$ and using $\sigma'$ we arrive in a proof for $\sigma'(f(\overrightarrow{x}_{ar(f)-1})) \xrightarrow{\sigma'(t'')}_r \sigma'(t')$ or $q \xrightarrow{\sigma'(t'')}_r \sigma'(t')$ and by construction of $\sigma'$, $p''\tilde{R}\sigma'(t'')$ and $p'\tilde{R}\sigma'(t')$. This concludes the proof item 1.

For the proof of item 3, consider a transition $p \xrightarrow{p''}_r p'$ such that $r$ is persistent for some $t''$ labels and $t = \sigma(t'')$. We show that for an arbitrary $\sigma'$ such that $\sigma$ and $\sigma'$ respect $\tilde{R}$, it holds that $p \xrightarrow{\sigma'(t'')}_r p''$ and $p'\tilde{R}p''$. Suppose that the statement holds for transitions with a proof of depth less than $n$ and consider a transition with a proof depth $n$.

Following Definition 9.15, the transition has to be due to a deduction rule of the following form:

$$\frac{\{t_i \xrightarrow{t_i'}_{r_i} y_i \mid i \in I\} \quad \{t_j \stackrel{t_j'}{\nrightarrow}_{r_j} \mid j \in J\}}{f(\overrightarrow{x}_{ar(f)-1}) \xrightarrow{s}_r t'}$$

and there should exist a substitution $\sigma''$ such that $p = \sigma''(f(\overrightarrow{x}_{ar(f)-1})$, $p'' = \sigma(t'') = \sigma(\sigma''(t''))$ and $p' = \sigma(t')$. As before, we aim at defining a new substitution $\alpha$. Take $\alpha_0$ to be defined as follows:

$$\alpha_0(x) = \begin{cases} \sigma'(\sigma''(x)) & x \in vars(t'') \\ \sigma(x) & x \notin vars(t'') \cup \{x_i, y_j \mid 0 \leq i < ar(f) \wedge j \in I\} \end{cases}$$

We complete the definition of $\alpha$ by adding the valuation of $y_i$ variables to $\alpha_0$ by exploiting the induction step and the persistency of transitions in the premises. Note that since $\sigma$ and $\sigma'$ respect $\tilde{R}$ and $\tilde{R}$ is a congruence then $\sigma \circ \sigma''$ and $\sigma' \circ \sigma''$ respect $\tilde{R}$, as well. Hence, $\sigma$ and $\alpha_0$ respect $\tilde{R}$ on their common domain. Using a similar procedure as before, we aim at completing the definition of $\alpha_0$ by following the chain of premises with respect to variable ordering $\leq_h$.

Take a premise of which all the variables of the source and labels are valuated by $\alpha_i$. Such a premise should exist due to acyclicity of variable dependency graph. Since $\sigma(t_j)\tilde{R}\alpha_i(t_j)$ and $\sigma(t_j')\tilde{R}\alpha_i(t_j')$, the induction hypothesis applies and there exists a $p_j'$ such that $\alpha_i(t_j) \xrightarrow{\alpha_i(t_j')}_{r_j} p_j'$. Then let $\sigma_{i+1}$ be defined as $\sigma_i[y_j \mapsto p_j']$.

Take $\alpha$ to be the union of all $\alpha_i$. It only remains to show that the negative premises of **(d)** hold when instantiated with $\alpha$. Suppose that there exists a premise $t_j \xrightarrow{t_j'}_{r_j}$ such that for some $p_j'$, it holds $\alpha(t_j) \xrightarrow{\alpha(t_j')}_{r_j} p_j'$. It also holds that $max(\mathcal{S}(\sigma(t_j), r_j, \sigma(t_j')), \mathcal{S}(\alpha(t_j), r_j, \alpha(t_j'))) \leq \mathcal{S}(\sigma(t_j), r_j, \sigma(t_j'))) < \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)-1})),$ $r_j, \sigma(t''))$). Hence, the induction hypothesis applies and $\sigma(t_j) \xrightarrow{\sigma(t_j')}_{r_j} p_j''$ for some $p_j''$, contradicting the provability of the transition of $p$.

In conclusion, using $\alpha$ and deduction rule **(d)**, we get a proof for $\alpha(t) \xrightarrow{\alpha t''}_r \alpha(t')$ and by construction of $\alpha$ it holds that $\sigma(t')\tilde{R}\alpha(t')$. ⊠

Now, we are in the position, to give a shorter proof for Theorem 9.16.

*Proof.* Rules defined by the hypotheses of Theorem 9.16 are in the higher order PANTH format and higher order bisimilarity is a congruence.

Next, we have to prove that $\underline{\leftrightarrow}_h = \underline{\leftrightarrow}_s$. It trivially holds that $\underline{\leftrightarrow}_s \subseteq \underline{\leftrightarrow}_h$, so, it remains to show that $\underline{\leftrightarrow}_h \subseteq \underline{\leftrightarrow}_s$. We prove that $\underline{\leftrightarrow}_h$ is a strong bisimulation relation by means of the following statement.

For all $p, q \in \mathcal{C}$ such that $p \underline{\leftrightarrow}_h q$ and for all $r \in Rel$ and $L \in \mathcal{L}$:

1. $\forall_{p' \in \mathcal{C}}\ p \xrightarrow{L}_r p' \Rightarrow \exists_{q' \in \mathcal{C}}\ q \xrightarrow{L}_r q' \wedge p' \underline{\leftrightarrow}_h q'$;
2. $\forall_{q' \in \mathcal{C}}\ q \xrightarrow{L}_r q' \Rightarrow \exists_{p' \in \mathcal{C}}\ p \xrightarrow{L}_r p' \wedge p' \underline{\leftrightarrow}_h q'$.

As before, it suffices to prove the transition condition for $p$ due to symmetry.

Since $p \underline{\leftrightarrow}_h q$, it follows from $p \xrightarrow{L}_r p'$ that $\exists_{L' \in \mathcal{L}}\ q \xrightarrow{L'}_r q'$ for some $q'$ such that $p' \underline{\leftrightarrow}_h q'$.

Since $L \underline{\leftrightarrow}_h L'$ then for a list $\overrightarrow{z}_n$ of variables (of the same size as $L$ and $L'$), there exists two substitutions $\sigma$ and $\sigma'$ such that $L = \sigma(\overrightarrow{z}_n)$ and $L' = \sigma'(\overrightarrow{z}_n)$ and $\sigma$ and $\sigma'$ respect $\underline{\leftrightarrow}_h$ . Thus, it follows from the proof of Theorem 9.20 that $q \xrightarrow{L}_r q''$ for some $q''$ such that $q' \underline{\leftrightarrow}_h q''$ (item 3 in the transfer conditions of the proof). It follows, then, by transitivity of the higher order bisimilarity that $p' \underline{\leftrightarrow}_h q''$. $\boxtimes$

**Example 9.22** (Congruence of Higher Order Bisimilarity for CHOCS) The semantics of CHOCS as given in Example 9.1 conforms to our format. To verify this claim we have to check that in the conclusion of all deduction rules mentions only one function symbol at a time, the target of premises mention distinct variables and the label of premises either mention distinct variables or are persistent. The first two checks are straightforward. For the third, the only problem arises from the rules having two premises mentioning the same label $z$. Three of such rules appear in the definition of substitution transitions which is shown to be persistent, so they conform to our format. The only other rule having the same condition is the one defining communication for parallel composition. But in that rule, the receive transition is persistent and hence, the only non-persistent premise (the send transition) trivially satisfies the second criterion of Definition 9.18. Note that the notion of higher order bisimilarity in [119] also requires that bisimilarity should be closed under substitution of atoms. Our notion does not require this in general, but in the case of CHOCS semantics, the addition of substitution, makes sure that bisimilar terms always have the same "substitution behavior". Hence, the two notions trivially coincide.

## 9.6 Conclusion

In this chapter, we presented two syntactic formats that guarantee congruence for two notions of strong and higher order bisimilarity. We applied these formats to the CHOCS process algebra [119].

Due to the abundant presence of notions of names and binders in the formalisms with higher-order behavior, the addition of these notions to our formats is a very natural and useful extension. We are currently considering this extension and we try to exploit the Gabbay-Pitts nominal techniques of [50, 104] for this purpose.

**Acknowledgment** Comments of the reviewers of the CONCUR conference are gratefully acknowledged. Alan Jeffrey, Soren Lassen and Paul Levy provided useful comments on this chapter leading to correction of Theorem 9.12.

# Chapter 10

# Conclusions

"Now, Earthlings ..." whirred the Vogan [...]
"I present you with a simple choice!
Either die in the vacuum of space, or ..."
He paused for a melodramatic effect.
"Tell me how good you thought my poem was! "

["The Hitchhiker's Guide to the Galaxy", Douglas Adams]

"An expert is a person who has made all the mistakes, which can be made, in a very narrow field."

[Niels Bohr]

"One never notices what has been done; one can only see what remains to be done."

[Maria Skłodowska-Curie]

In this thesis, we presented an overview of Structural Operational Semantics (SOS) and its formal frameworks in terms of a Transition System Specifications (TSS's). We also reviewed existing meta-results about TSS's. Subsequently, we made the following contributions to these frameworks and meta-results.

1. A commutativity meta-theorem was presented which guarantees that some function symbols in the signature are commutative with respect to strong bisimilarity.

2. The operational semantic specification was extended with a set of equational specifications (called structural congruences) and meta-theorems concerning congruence of bisimilarity and well-definedness of the semantics were reformulated in the extended setting.

3. Novel and more liberal notions for operational and equational conservativity were introduced and some meta-results around them were presented.

4. A prototype version of an SOS toolset was implemented and reported. This prototype allows for checking simple instances of congruence and conservativity meta-theorems and provides the possibility of animating SOS specifications.

5. Existing SOS frameworks were extended to the setting with an explicit data part. Notions of bisimilarity with data were studied and congruence formats for them were proposed.

6. A TSS framework for higher order processes was presented and congruence meta-theorems for strong and higher-order bisimilarities were presented.

A lot remains to be done in this area. At the end of each chapter, we listed a number of possible extensions to the contributions of the chapter. Among those, the following items are our first priorities.

1. Studying the notions of names and binders. We see the nominal techniques of Gabbay and Pitts [50, 104] as a very convenient departure point. Most of the meta-results presented in this thesis can be extended with these concepts. As a distinguished example, there is no congruence meta-theorem for strong bisimilarity for TSS's with binders and terms as labels. If we succeed in our study, this framework will be the top element in the lattice of frameworks presented in Chapter 3;

2. Studying the notions of congruence for bisimulation with data and higher-order bisimulation in the bi-algebraic framework of Turi and Plotkin [123];

3. Turning our prototype into a full-fledged toolset for assisting language designers.

# References


[1] The Maude system. Available from `http://maude.cs.uiuc.edu/`.

[2] Luca Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In Bengt Jonsson and Joachim Parrow, editors, *Proceedings of the fifth International Conference on Concurrency Theory (CONCUR'94)*, volume 836 of *Lecture Notes in Computer Science*, pages 449–464. Springer-Verlag, Berlin, Germany, 1994.

[3] Luca Aceto, Bard Bloom, and Frits W. Vaandrager. Turning SOS rules into equations. *Information and Computation (I&C)*, 111:1–52, 1994.

[4] Luca Aceto, Willem Jan (Wan) Fokkink, and Chris Verhoef. Conservative extension in structural operational semantics. In Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Current Trends in Theoretical Computer Science - Entering the 21st Century*, pages 504–524. World Scientific, Singapore, 2001.

[5] Luca Aceto, Willem Jan (Wan) Fokkink, and Chris Verhoef. Structural operational semantics. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra, Chapter 3*, pages 197–292. Elsevier Science, Dordrecht, The Netherlands, 2001.

[6] Egidio A Astesiano, Alessandro Giovini, and Gianna Reggio. Generalized bisimulation in relational specifications. In Robert Cori and Martin Wirsing, editors, *Proceedings of the 5th Annual Symposium on Theoretical Aspects of Computer Science (STACS'88)*, volume 294 of *Lecture Notes in Computer Science*, pages 207–226. Springer-Verlag, Berlin, Germany, 1988.

[7] J.C.M. (Jos) Baeten and Jan A. Bergstra. Processen en procesexpressies. *Informatie*, 30(3):177–248, 1988. (In Dutch).

[8] J.C.M. (Jos) Baeten and Jan A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2):188–208, 1996.

[9] J.C.M. (Jos) Baeten and Cornelis A. (Kees) Middelburg. *Process Algebra with Timing*. EATCS Monographs. Springer-Verlag, Berlin, Germany, 2002.

[10] J.C.M. (Jos) Baeten, MohammadReza Mousavi, and Michel A. Reniers. Timing the untimed: Terminating successfully while being conservative. Technical Report 05-21, Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2005.

[11] J.C.M. (Jos) Baeten and Chris Verhoef. Concrete Process Algebra. In Samson Abramsky, Dov M. Gabbay, and Thomas S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, *Semantic Modelling*, pages 149–268. Oxford University Press, 1995.

[12] J.C.M. (Jos) Baeten and Chris Verhoef. A congruence theorem for structured operational semantics with predicates. In Eike Best, editor, *International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 477–492. Springer-Verlag, Berlin, Germany, 1993.

[13] J.C.M. (Jos) Baeten and W. Peter Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambrdige University Press, 1990.

[14] Jos C. M. Baeten. Embedding untimed into timed process algebra: the case for explicit termination. *Mathematical Structures in Computer Science (MSCS)*, 13(4):589–618, 2003.

[15] Jos C.M. Baeten and Erik P. de Vink. Axiomatizing GSOS with termination. In Helmut Alt and Afonso Ferreira, editors, *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2295 of *Lecture Notes in Computer Science*, pages 583–595. Springer-Verlag, Berlin, Germany, 2002.

[16] Jos C.M. Baeten and Erik P. de Vink. Axiomatizing GSOS with termination. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:323–351, 2004.

[17] Jaco W. de Bakker and Erik P. de Vink. *Control Flow Semantics*. Foundations of Computing Series. The MIT Press, 1996.

[18] Jean-Pierre Banâtre, Pascal Fradet, and Daniel Le Métayer. Gamma and the chemical reaction model: Fifteen years after. In Cristian S. Calude, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Multiset Processing: Mathematical, Computer Science, and Molecular Computing Points of View*, volume 2235 of *Lecture Notes in Computer Science*, pages 17–44. Springer-Verlag, Berlin, Germany, 2001.

[19] Falk Bartels. GSOS for probabilistic transition systems. In *Proceedings of the 5th International Workshop on Coalgebraic Methods in Computer Science (CMCS'02)*, volume 65 of *Electronic Notes in Theoretical Computer Science*, pages 1–25, 2002.

[20] Jan A. Bergstra and Cornelis A. (Kees) Middelburg. Process algebra for hybrid systems. *Theoretical Computer Science*, 335(2-3):215–280, 2005.

[21] Karen L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *IEEE Symposium on Logic In Computer Science (LICS'98)*, pages 153–164. IEEE Computer Society, Los Alamitos, CA, USA, 1998.

[22] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science (TCS)*, 96:217–248, 1992.

[23] Bard Bloom. Structural operational semantics for weak bisimulations. *Theoretical Computer Science (TCS)*, 146:25–68, 1995.

[24] Bard Bloom, Willem Jan (Wan) Fokkink, and Robert Jan (Rob) van Glabbeek. Precongruence formats for decorated trace semantics. *ACM Transactions on Computational Logic*, 5(1):26–78, 2004.

[25] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM (JACM)*, 42(1):232–268, January 1995.

[26] Bard Bloom and Frits W. Vaandrager. SOS rule formats for parameterized and state-bearing processes (draft). Unpublished note, available through: `http://www.cs.kun.nl/ita/publications/papers/fvaan/`.

[27] Roland Bol and Jan Friso Groote. The meaning of negative premises in transition system specifications. *Journal of the ACM (JACM)*, 43(5):863–914, September 1996.

[28] Patrick Borras, Dominique Clément, Thierry Despeyroux, Janet Incerpi, Gilles Kahn, Bernard Lang, and Valérie Pascual. CENTAUR: The system. *SIGPLAN Notices*, 24(2):14–24, 1988.

[29] Victor Bos and Jeroen J.T. Kleijn. Redesign of a systems engineering language — formalisation of $\chi$. *Formal Aspects of Computing*, 15(4), December 2003.

[30] Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In Josep Díaz and Fernando Orejas, editors, *Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT'89)*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161. Springer-Verlag, Berlin, Germany, 1989.

[31] Christiano de O. Braga. *Rewriting Logic as a Semantic Framework for Modular Structural Operational Semantics*. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica de Rio de Janeiro, Brasil, 2001. `http://www.ic.uff.br/~cbraga`.

[32] Christiano de O. Braga, Edward Hermann Haeusler, José Meseguer, and Peter D. Mosses. Mapping modular SOS to rewriting logic. In Michael Leuschel, editor, *Proceedings of the 12th International Workshop on Logic Based Program Synthesis and Tranformation (LOPSTR'02)*, volume 2664 of *Lecture Notes in Computer Science*, pages 262–277. Springer-Verlag, Berlin, Germany, 2002.

[33] Christiano de O. Braga, Edward Hermann Haeusler, José Meseguer, and Peter D. Mosses. Maude action tool: Using reflection to map action semantics to rewriting logic. In Teodor Rus, editor, *Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology (AMAST'00)*, volume 1816 of *Lecture Notes in Computer Science*, pages 407–421. Springer-Verlag, Berlin, Germany, 2000.

[34] Michel Bréal. *Semantics: Studies in the science of meaning*. Dover Publications Inc., New York, USA, 1964.

[35] Antonio Brogi and Jean-Marie Jacquet. On the expressiveness of Linda-like concurrent languages. In Ilaria Castellani and Catuscia Palamidessi, editors, *Proceedings of the 5th International Workshop on Expressiveness in Concurrency (EXPRESS'98)*, volume 16 of *Electronic Notes in Theoretical Computer Science (ENTCS)*. Elsevier Science, Dordrecht, The Netherlands, 1998.

[36] Roberto Bruni, Ugo Montanari, and Vladimiro Sassone. Observational congruences for dynamically reconfigurable tile systems. *Theoretical Computer Science (TCS)*, 335(2-3):331–372, 2005.

[37] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science (TCS)*, 240(1):177–213, 2000.

[38] Michel R. V. Chaudron. *Separating Computation and Coordination in the Design of Parallel and Distributed Programs*. PhD thesis, Department of Computer Science, Rijksuniversiteit Leiden, Leiden, The Netherlands, 1998.

[39] Michel R. V. Chaudron and Edwin de Jong. Schedules for multiset transformer programs. In *Coordination Programming: Mechanisms, Models and Semantics*, pages 195–210. Imperial College Press, London, UK, 1996.

[40] Dominique Clément, Janet Incerpi, and Gilles Kahn. CENTAUR: Towards a "software tool box" for programming environments. In Fred Long, editor, *Proceedings of the International Workshop on Software Engineering*

*Environments (SEE'89)*, volume 467 of *Lecture Notes in Computer Science*, pages 287–304. Springer-Verlag, Berlin, Germany, 1990.

[41] Pieter J.L. Cuijpers and Michel A. Reniers. Hybrid process algebra. *Journal of Logic and Algebraic Programming (JLAP)*, 62(2):191–245, 2005.

[42] Robert de Simone. Higher-level synchronizing devices in MEIJE-SCCS. *Theoretical Computer Science (TCS)*, 37:245–267, 1985.

[43] Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. A causal semantics for CCS via rewriting logic. *Theoretical Computer Science*, 275(1-2):259–282, 2002.

[44] Joost Engelfriet and Tjalling Galsema. Multisets and structutal congruence of pi-calculus with replication. *Acta Informatica*, 211(2):311–337, 2004.

[45] Willem Jan (Wan) Fokkink. The tyft/tyxt format reduces to tree rules. In Masami Hagiya and John C. Mitchell, editors, *Proceedings of the Symposium on Theoretical Aspects of Computer Software (STACS'94)*, volume 789 of *Lecture Notes in Computer Science*, pages 440–453. Springer-Verlag, Berlin, Germany, 1994.

[46] Willem Jan (Wan) Fokkink and Robert Jan (Rob) van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation (I&C)*, 126(1):1–10, 1996.

[47] Willem Jan (Wan) Fokkink, Robert Jan (Rob) van Glabbeek, and Paulien de Wind. Compositionality of Hennessy-Milner logic through structural operational semantics. In Andrzej Lingas and Bengt J. Nilsson, editors, *Proceedings of the 14th Symposium on Fundamentals of Computation Theory (FCT'03)*, volume 2751 of *Lecture Notes in Computer Science*, pages 412–422. Springer-Verlag, Berlin, Germany, 2003.

[48] Willem Jan (Wan) Fokkink and Chris Verhoef. A conservative look at operational semantics with variable binding. *Information and Computation (I&C)*, 146(1):24–54, 1998.

[49] Willem Jan (Wan) Fokkink and Thuy Duong Vu. Structural operational semantics and bounded nondeterminism. *Acta Informatica*, 39(6–7):501–516, 2003.

[50] Murdoch J. Gabbay and James Cheney. A sequent calculus for nominal logic. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, pages 139–148. IEEE Computer Science, Los Alamitos, CA, USA, 2004.

[51] Fabio Gadducci and Ugo Montanari. The tile model. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 133–166. MIT Press, Boston, MA, USA, 2000.

[52] Vashti Galpin. A format for semantic equivalence comparison. *Theoretical Computer Science (TCS)*, 309(1-3):65–109, 2003.

[53] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, Cambridge, MA, USA, 1988.

[54] Robert Jan (Rob) van Glabbeek. Bounded nondeterminism and the approximation induction principle in process algebra. In Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS'87)*, volume 247 of *Lecture Notes in Computer Science*, pages 336–347. Springer-Verlag, Berlin, Germany, 1987.

[55] Robert Jan (Rob) van Glabbeek. Full abstraction in structural operational semantics (extended abstract). In Maurice Nivat, Charles Rattray, Teodor Rus, and Giuseppe Scollo, editors, *Proceedings of the Third International Conference on Algebraic Methodology and Software Technology (AMAST '93)*, pages 75–82. Springer-Verlag, Berlin, Germany, 1994.

[56] Robert Jan (Rob) van Glabbeek. The linear time - branching time spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra, Chapter 1*, pages 3–100. Elsevier Science, Dordrecht, The Netherlands, 2001.

[57] Robert Jan (Rob) van Glabbeek. The linear time - branching time spectrum II. In Eike Best, editor, *International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, Berlin, Germany, 1993.

[58] Robert Jan (Rob) van Glabbeek. On cool congruence formats for weak bisimulations (extended abstract). In *Proceedings of the 2nd International Colloquium on Theoretical Aspects of Computing (ICTAC'05)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2005. To appear.

[59] Robert Jan (Rob) van Glabbeek. The meaning of negative premises in transition system specifications II. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:229–258, 2004.

[60] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, Los Alamitos, CA, USA, 1982.

[61] Jan Friso Groote. Transition system specifications with negative premises. *Theoretical Computer Science (TCS)*, 118(2):263–299, 1993.

[62] Jan Friso Groote. The syntax and semantics of timed $\mu CRL$. Technical Report SEN-R9709, CWI - Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, June 30, 1997.

[63] Jan Friso Groote and Alban Ponse. Process algebra with guards: Combining Hoare logic with process algebra. *Formal Aspects of Computing (FAC)*, 6(2):115–164, 1994.

[64] Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation (I&C)*, 100(2):202–260, October 1992.

[65] Pieter H. Hartel. LETOS - a lightweight execution tool for operational semantics. *Software, Practice and Experience*, 29(15):1379–1416, 1999.

[66] Matthew Hennessy and Gordon D. Plotkin. Full abstraction for a simple parallel programming language. In Jirí Becvár, editor, *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science (MFCS'79)*, volume 74 of *Lecture Notes in Computer Science*, pages 108–120. Springer-Verlag, Berlin, Germany, 1979.

[67] Matthew C. B. Hennessy and A.J.R.G. (Robin) Milner. Algebraic laws for non-determinism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[68] C.A.R. (Tony) Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[69] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science (TCS)*, (2):437–486, 1995.

[70] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation (I&C)*, 124:103–112, 1996.

[71] Jean-Marie Jacquet, Koenraad De Bosschere, and Antonio Brogi. On timed coordination languages. In António Porto and Gruia-Catalin Roman, editors, *Proceedings of the 4th International Conference on Coordination Languages and Models (COORDINATION'00)*, volume 1906 of *Lecture Notes in Computer Science*, pages 81–98. Springer-Verlag, Berlin, Germany, 2000.

[72] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation (I& C)*, 86(1):43–68, 1990.

[73] Ruggero Lanotte and Simone Tini. Probabilistic congruence for semistochastic generative processes. In Vladimiro Sassone, editor, *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 63–78. Springer-Verlag, 2005.

[74] Kim G. Larsen. *Context-Dependent Bisimulation between Processes*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, UK, 1986.

[75] Kim G. Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.

[76] Guy Leduc and Luc Leonard. A timed LOTOS supporting a dense time domain and including new timed operators. In Michael Diaz and Roland Groz, editors, *Proceedings of the Fifth International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'92)*, volume C-10 of *IFIP Transactions*, pages 87–102. North-Holland, 1993.

[77] James J. Leifer. *Operational Congruences for Reactive Systems*. PhD thesis, Computer Laboratory, University of Cambridge, Cambridge, UK, 2001.

[78] James J. Leifer and A.J.R.G. (Robin) Milner. Deriving bisimulation congruences for reactive systems. In Catuscia Palamidessi, editor, *Proceedings of 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 259–274. Springer-Verlag, Berlin, Germany, 2000.

[79] Sebastian P. Luttik. *Choice quantification in process algebra*. PhD thesis, Department of Computer Science, University of Amsterdam, Amsterdam, The Netherlands, 2002.

[80] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic*, volume 9, pages 1–87. Kluwer Academic Publishers, 2002.

[81] José Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theoretical Computer Science (TCS)*, 96(1):73–155, 1992.

[82] José Meseguer and Christiano Braga. Modular rewriting semantics of programming languages. In Charles Rattray, Savi Maharaj, and Carron Shankland, editors, *Proceedings of the 10th International Conference on Algebraic Methodology and Software Technology (AMAST'04)*, volume 3116 of *Lecture Notes in Computer Science*, pages 364–378. Springer-Verlag, Berlin, Germany, 2004.

[83] Cornelis A. (Kees) Middelburg. Variable binding operators in transition system specifications. *Journal of Logic and Algebraic Programming*, 47(1):15–45, 2001.

[84] Cornelis A. (Kees) Middelburg. An alternative formulation of operational conservativity with binding terms. *Journal of Logic and Algebraic Programming (JLAP)*, 55(1-2):1–19, 2003.

[85] A.J.R.G. (Robin) Milner. Flowgraphs and flow algebras. *Journal of the ACM (JACM)*, 26(2):794–818, 1979.

[86] A.J.R.G. (Robin) Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[87] A.J.R.G. (Robin) Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[88] A.J.R.G. (Robin) Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119–141, 1992. An earlier version of this paper appeared as Technical Report N.1154 of INRIA, Sophia-Antipolis, 1990.

[89] A.J.R.G. (Robin) Milner. The polyadic $\pi$-calculus: a tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993. An earlier version of this paper appeared as Technical Report ECS-LFCS-91-180 of University of Edinburgh, 1991.

[90] A.J.R.G. (Robin) Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I. *Information and Computation (I&C)*, 100(1):1–40, 1992. An earlier version of this paper appeared as Technical Report ECS-LFCS-89-85 of University of Edinburgh, 1989.

[91] A.J.R.G. (Robin) Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part II. *Information and Computation (I&C)*, 100(1):41–77, 1992. An earlier version of this paper appeared as Technical Report ECS-LFCS-89-86 of University of Edinburgh, 1989.

[92] A.J.R.G. (Robin) Milner and Davide Sangiorgi. Barbed bisimulation. In Werner Kuich, editor, *Proceedings of 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, volume 623 of *Lecture Notes in Computer Science*, pages 85–695. Springer-Verlag, Berlin, Germany, 1992.

[93] Peter D. Mosses. Exploiting labels in structural operational semantics. *Fundamenta Informaticae*, 60:17–31, 2004.

[94] Peter D. Mosses. Modular structural operational semantics. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:195–228, 2004.

[95] MohammadReza Mousavi, Twan Basten, Michel Reniers, Michel Chaudron, and Giovanni Russello. Separating functionality, behavior and timing in the design of reactive systems: (GAMMA + coordination) + time. Technical Report 02-09, Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2002.

[96] MohammadReza Mousavi, Murdoch J. Gabbay, and Michel A. Reniers. SOS for higher order processes. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2005. To appear.

[97] MohammadReza Mousavi and Michel Reniers. Structural congruences and structural operational semantics. Technical Report CSR-04-28, Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, October 2004.

[98] MohammadReza Mousavi, Michel Reniers, and Jan Friso Groote. A syntactic commutativity format for SOS. *Information Processing Letters (IPL)*, 93:217–223, March 2005.

[99] MohammadReza Mousavi, Marjan Sirjani, and Farhad Arbab. Specification and verification of component connectors. Technical Report CSR-04-15, Department of Computer Science, Eindhoven University of Technology, 2004.

[100] Xavier Nicollin and Joseph Sifakis. The algebra of timed processes ATP: theory and application. *Information and Computation (I&C)*, 114(1):131–178, October 1994.

[101] Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[102] David M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, Berlin, Germany, 1981.

[103] Iain Phillips and Irek Ulidowski. Ordered SOS rules and weak bisimulation. In Abbas Edalat, Sofia Jourdan, and Guy McCusker, editors, *Advances in Theory and Formal Methods of Computing*, pages 300–311. Imperial College Press, London, UK, 1996.

[104] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation (I&C)*, 186(2):165–193, 2003.

[105] Gordon D. Plotkin. An operational semantics for CSO. In Andrzej Salwicki, editor, *Proceedings of the Conference on Logic of Programs and Their Applications (1980)*, volume 148 of *Lecture Notes in Computer Science*, pages 250–252. Springer-Verlag, Berlin, Germany, 1983.

[106] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, September 1981.

[107] Gordon D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming (JLAP)*, 60:3–15, 2004.

[108] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Progamming (JLAP)*, 60:17–139, 2004. This article first appeared as [106].

[109] Michel A. Reniers, Jan Friso Groote, Mark B. van der Zwaag, and Jos van Wamel. Completeness of timed $\mu CRL$. *Fundamenta Informaticae*, 50(3-4):361–402, 2002.

[110] Arend Rensink. Bisimilarity of open terms. *Information and Computation (I&C)*, 156:345–385, 2000.

[111] Jan J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science (TCS)*, 249(1):3–80, 2000.

[112] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[113] David Sands. From SOS rules to proof principles: An operational metatheory for functional languages. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97)*, pages 428–441. ACM Press, New York, 1997.

[114] Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation (I& C)*, 111:120–153, 1994.

[115] Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation (I&C)*, 131(2):141–178, 1996.

[116] Vladimiro Sassone and Paweł Sobociński. Deriving bisimulation congruences: 2-categories vs precategories. In Andrew Gordon, editor, *Proceedings of the 6th International Conference on Foundations of Software Science and Computation Structures (FOSSACS '03)*, volume 2620 of *Lecture Notes in Computer Science*, pages 409–424. Springer-Verlag, Berlin, Germany, 2003.

[117] Peter Sewell. From rewrite rules to bisimulation congruences. *Theoretical Computer Science (TCS)*, 274(1-2):183–230, 2002.

[118] Bent Thomsen. Plain CHOCS a second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[119] Bent Thomsen. A theory of higher order communicating systems. *Information and Computation (I& C)*, 116:38–57, 1995.

[120] Simone Tini. Rule formats for non interference. In Pierpaolo Degano, editor, *Proceedings of the 12th European Symposium on Programming, Programming Languages and Systems (ESOP'03)*, volume 2618 of *Lecture Notes in Computer Science*, pages 129–143. Springer-Verlag, Berlin, Germany, 2003.

[121] Simone Tini. Rule formats for compositional non-interference properties. *Journal of Logic and Algebraic Progamming (JLAP)*, 60:353–400, 2004.

[122] Daniele Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University, Amsterdam, June 1996.

[123] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proceedings of the 12th Annual IEEE Symposium on Logic in computer Science (LICS'97)*, pages 280–291. IEEE Computer Society Press, 1997.

[124] David A. Turner. Miranda: a non-strict functional language with polymorphic types. In Jean-Pierre Jouannaud, editor, *Proceeding of the ACM Conference on Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, Germany, 1985.

[125] Irek Ulidowski. Finite axiom systems for testing preorder and De Simone process languages. *Theoretical Computer Science (TCS)*, 239(1):97–139, 2000.

[126] Irek Ulidowski and Iain Phillips. Ordered SOS process languages for branching and eager bisimulations. *Information and Computation (I&C)*, 178(1):180–213, 2002.

[127] Frits W. Vaandrager. Expressiveness results for process algebras. In Jaco W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Proceedings of the REX Workshop on Semantics*, volume 666 of *Lecture Notes in Computer Science*, pages 609–638. Springer-Verlag, Berlin, Germany, 1993.

[128] Frits W. Vaandrager. On the relationship between process algebra and input/output automata. In *Proceedings of the Sixth Annual IEEE Symposium onLogic in Computer Science (LICS'91)*, pages 387–398. IEEE Computer Society, 1991.

[129] Alberto Verdejo. Building tools for LOTOS symbolic semantics in Maude. In Doron Peled and Moshe Vardi, editors, *Proceedings of the 22nd IFIP International Conference on Formal Techniques for Networked and Distributed Systenms (FORTE'02)*, volume 2529 of *Lecture Notes in Computer Science*, pages 292–307. Springer-Verlag, Berlin, Germany, 2002.

[130] Alberto Verdejo and Narciso Martí-Oliet. Implementing CCS in Maude. In Tommaso Bolognesi and Diego Latella, editors, *Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX)*, volume 183 of *IFIP Conference Proceedings*, pages 351–366. Kluwer Academic Publishers, 2000.

[131] Alberto Verdejo and Narciso Martí-Oliet. Implementing CCS in Maude 2. In Fabio Gadducci and Ugo Montanari, editors, *Proceedings of the 4th International Workshop on Rewriting Logic and its Applications (WRLA'02)*, volume 71 of *Electronic Notes on Theoretical Computer Science (ENTCS)*, pages 239–257. Elsevier Science, Dordrecht, The Netherlands, 2002.

[132] Alberto Verdejo and Narciso Martí-Oliet. Executable structural operational semantics in Maude. Technical Report 134-03, Dpto. Sistemas Informáticos y Programación, Universidad Complutense de Madrid, September 2003. `http://www.ucm.es/info/dsip/alberto/papers/JLAP-TR.html`.

[133] Jan Joris Vereijken. *Discrete Time Process Algebra*. PhD thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1997.

[134] Chris Verhoef. A general conservative extension theorem in process algebra. In Ernst-Rüdiger Olderog, editor, *Proceedings of third IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94)*, volume A-56 of *IFIP Transactions*, pages 274–302. Elsevier Science Publishers, 1994.

[135] Chris Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2(2):274–302, 1995.

[136] Yingzhou Zhang and Baowen Xu. A survey of semantic description frameworks for programming languages. *SIGPLAN Notices*, 39(3):14–30, 2004.

# Summary

Defining a formal (i.e., mathematical) semantics for computer languages is the first step towards developing rigorous techniques for reasoning about computer programs and specifications in such a language. Structural Operational Semantics (SOS), introduced by Plotkin in 1981, has become a popular technique for defining formal semantics. In this thesis, we first review the basic concepts of SOS and the existing meta-results. Subsequently, we enhance the state of the art in this field by offering the following contributions:

- developing a syntactic format guaranteeing a language construct to be commutative;
- extending the existing congruence and well-definedness meta-results to the setting with equational specifications;
- defining a more liberal notion of operational conservativity, called orthogonality, and formulating meta-theorems for it;
- prototyping a framework for checking the premises of congruence and conservativity meta-theorems and animating programs according to their SOS specification;
- defining notions of bisimulation with data and formulating syntactic rule formats guaranteeing congruence for these notions;
- proposing syntactic rule formats for guaranteeing congruence of strong bisimilarity and higher-order bisimilarity in the setting of higher order processes.

# Samenvatting

De beschrijving van een formele (i.e., wiskundige) semantiek voor een computer-gerelateerde taal is de eerste stap naar ontwikkeling van nieuwe technieken voor het redeneren over computerprogramma's en specificaties in zo'n taal. Structurele Operationele Semantiek (SOS), geïntroduceerd door Plotkin in 1981, is een populaire methode voor het beschrijven van formele semantiek. In dit proefschrift bestuderen we eerst de fundamentele concepten van SOS en de huidige resultaten daarover. Vervolgens verbeteren we de stand van zaken door middel van de onderstaande bijdragen tot dit gebied:

- ontwikkeling van een syntactisch formaat dat garandeert dat een taalconstructie commutatief is;
- uitbreiding van de huidige congruentie en wel-gedefinieerdheid meta-stellingen met equationele beschrijvingen;
- definiëren van een liberale notie van conservatieve uitbreiding, genaamd orthogonaliteit, en het formuleren van meta-stellingen;
- het prototyperen van een raamwerk ter verificatie van de voorwaarden van congruentie en conservativiteits meta-stellingen en het animeren van programma's volgens hun SOS beschrijvingen;
- definiëren van noties van bisimulatie met data en het formuleren van formaten voor congruentie;
- voorstellen van syntactische formaten voor congruentie van sterke en hogere order bisimulatie voor hogere order processen.

# Curriculum Vitae

MohammadReza Mousavi was born on the 3rd of July 1978 in Tehran, Iran. In 1995, he graduated from the Allame-Helli High School in Mathematics and Physics. He received his Bachelor and Masters degrees, both in Computer Engineering (Software), from Sharif University of Technology, Tehran, Iran in 1999 and 2001, respectively. For his Masters degree he received the best students award in 2001. Since October 2001, he has been a Ph.D. student within the Computer Science Department of the Technische Universiteit Eindhoven (TU/e) in Eindhoven, The Netherlands. His research was funded by NWO (The Dutch Organization for Scientific Research) within the project SACC: Software Architecture = Components + Coordination. His research has led amongst others to several publications at international conferences and in journals, and to this thesis. Mohammad currently lives in Eindhoven and as of October 2005, he is employed for two years by TU/e as an assistant professor in Computer Science and Electrical Engineering.

# Index

## Titles in the IPA Dissertation Series

**J.O. Blanco**. *The State Operator in Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1996-01

**A.M. Geerling**. *Transformational Development of Data-Parallel Algorithms.* Faculty of Mathematics and Computer Science, KUN. 1996-02

**P.M. Achten**. *Interactive Functional Programs: Models, Methods, and Implementation.* Faculty of Mathematics and Computer Science, KUN. 1996-03

**M.G.A. Verhoeven**. *Parallel Local Search.* Faculty of Mathematics and Computing Science, TUE. 1996-04

**M.H.G.K. Kesseler**. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory.* Faculty of Mathematics and Computer Science, KUN. 1996-05

**D. Alstein**. *Distributed Algorithms for Hard Real-Time Systems.* Faculty of Mathematics and Computing Science, TUE. 1996-06

**J.H. Hoepman**. *Communication, Synchronization, and Fault-Tolerance.* Faculty of Mathematics and Computer Science, UvA. 1996-07

**H. Doornbos**. *Reductivity Arguments and Program Construction.* Faculty of Mathematics and Computing Science, TUE. 1996-08

**D. Turi**. *Functorial Operational Semantics and its Denotational Dual.* Faculty of Mathematics and Computer Science, VUA. 1996-09

**A.M.G. Peeters**. *Single-Rail Handshake Circuits.* Faculty of Mathematics and Computing Science, TUE. 1996-10

**N.W.A. Arends**. *A Systems Engineering Specification Formalism.* Faculty of Mechanical Engineering, TUE. 1996-11

**P. Severi de Santiago**. *Normalisation in Lambda Calculus and its Relation to Type Inference.* Faculty of Mathematics and Computing Science, TUE. 1996-12

**D.R. Dams**. *Abstract Interpretation and Partition Refinement for Model Checking.* Faculty of Mathematics and Computing Science, TUE. 1996-13

**M.M. Bonsangue**. *Topological Dualities in Semantics.* Faculty of Mathematics and Computer Science, VUA. 1996-14

**B.L.E. de Fluiter**. *Algorithms for Graphs of Small Treewidth.* Faculty of Mathematics and Computer Science, UU. 1997-01

**W.T.M. Kars**. *Process-algebraic Transformations in Context.* Faculty of Computer Science, UT. 1997-02

**P.F. Hoogendijk**. *A Generic Theory of Data Types.* Faculty of Mathematics and Computing Science, TUE. 1997-03

**T.D.L. Laan**. *The Evolution of Type Theory in Logic and Mathematics.* Faculty of Mathematics and Computing Science, TUE. 1997-04

**C.J. Bloo**. *Preservation of Termination for Explicit Substitution.* Faculty of Mathematics and Computing Science, TUE. 1997-05

**J.J. Vereijken**. *Discrete-Time Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1997-06

**F.A.M. van den Beuken**. *A Functional Approach to Syntax and Typing.* Faculty of Mathematics and Informatics, KUN. 1997-07

**A.W. Heerink**. *Ins and Outs in Refusal Testing.* Faculty of Computer Science, UT. 1998-01

**G. Naumoski and W. Alberts**. *A Discrete-Event Simulator for Systems Engineering.* Faculty of Mechanical Engineering, TUE. 1998-02

**J. Verriet**. *Scheduling with Communication for Multiprocessor Computation.* Faculty of Mathematics and Computer Science, UU. 1998-03

**J.S.H. van Gageldonk**. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

**A.A. Basten**. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

**E. Voermans**. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

**H. ter Doest**. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

**J.P.L. Segers**. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

**C.H.M. van Kemenade**. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

**E.I. Barakova**. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

**M.P. Bodlaender**. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

**M.A. Reniers**. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

**J.P. Warners**. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

**J.M.T. Romijn**. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

**P.R. D'Argenio**. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

**G. Fábián**. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

**J. Zwanenburg**. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

**R.S. Venema**. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

**J. Saraiva**. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

**R. Schiefer**. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

**K.M.M. de Leeuw**. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

**T.E.J. Vos**. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

**W. Mallon**. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

**W.O.D. Griffioen**. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

**P.H.F.M. Verhoeven**. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

**J. Fey**. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

**M. Franssen**. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

**P.A. Olivier**. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

**E. Saaman**. *Another Formal Specification Language.* Faculty of Mathematics and Natural Sciences, RUG. 2000-10

**M. Jelasity**. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure.* Faculty of Mathematics and Natural Sciences, UL. 2001-01

**R. Ahn**. *Agents, Objects and Events a computational approach to knowledge, observation and communication.* Faculty of Mathematics and Computing Science, TU/e. 2001-02

**M. Huisman**. *Reasoning about Java programs in higher order logic using PVS and Isabelle.* Faculty of Science, KUN. 2001-03

**I.M.M.J. Reymen**. *Improving Design Processes through Structured Reflection.* Faculty of Mathematics and Computing Science, TU/e. 2001-04

**S.C.C. Blom**. *Term Graph Rewriting: syntax and semantics.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

**R. van Liere**. *Studies in Interactive Visualization.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

**A.G. Engels**. *Languages for Analysis and Testing of Event Sequences.* Faculty of Mathematics and Computing Science, TU/e. 2001-07

**J. Hage**. *Structural Aspects of Switching Classes.* Faculty of Mathematics and Natural Sciences, UL. 2001-08

**M.H. Lamers**. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes.* Faculty of Mathematics and Natural Sciences, UL. 2001-09

**T.C. Ruys**. *Towards Effective Model Checking.* Faculty of Computer Science, UT. 2001-10

**D. Chkliaev**. *Mechanical verification of concurrency control and recovery protocols.* Faculty of Mathematics and Computing Science, TU/e. 2001-11

**M.D. Oostdijk**. *Generation and presentation of formal mathematical documents.* Faculty of Mathematics and Computing Science, TU/e. 2001-12

**A.T. Hofkamp**. *Reactive machine control: A simulation approach using $\chi$.* Faculty of Mechanical Engineering, TU/e. 2001-13

**D. Bošnački**. *Enhancing state space reduction techniques for model checking.* Faculty of Mathematics and Computing Science, TU/e. 2001-14

**M.C. van Wezel**. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

**V. Bos and J.J.T. Kleijn**. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

**T. Kuipers**. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

**S.P. Luttik**. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

**R.J. Willemen**. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

**M.I.A. Stoelinga**. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

**N. van Vugt**. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

**A. Fehnker**. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

**R. van Stee**. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

**D. Tauritz**. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

**M.B. van der Zwaag**. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

**J.I. den Hartog**. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

**L. Moonen**. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

**J.I. van Hemert**. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

**S. Andova**. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

**Y.S. Usenko**. *Linearization in $\mu$CRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

**J.J.D. Aerts**. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

**M. de Jonge**. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

**J.M.W. Visser**. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

**S.M. Bohte**. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

**T.A.C. Willemse**. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

**S.V. Nedea**. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

**M.E.M. Lijding**. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

**H.P. Benz**. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

**D. Distefano**. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

**M.H. ter Beek**. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

**D.J.P. Leijen**. *The $\lambda$ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

**W.P.A.J. Michiels**. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

**G.I. Jojgov**. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

**P. Frisco**. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of

Mathematics and Natural Sciences, UL. 2004-03

**S. Maneth**. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

**Y. Qian**. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

**F. Bartels**. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

**L. Cruz-Filipe**. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

**E.H. Gerding**. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

**N. Goga**. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

**M. Niqui**. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

**A. Löh**. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

**I.C.M. Flinsenberg**. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

**R.J. Bril**. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

**J. Pang**. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

**F. Alkemade**. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

**E.O. Dijk**. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

**S.M. Orzan**. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

**M.M. Schrage**. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

**E. Eskenazi and A. Fyukov**. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

**P.J.L. Cuijpers**. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

**N.J.M. van den Nieuwelaar**. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

**E. Ábrahám**. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

**R. Ruimerman**. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

**C.N. Chong**. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

**H. Gao**. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

**H.M.A. van Beek**. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

**M.T. Ionita**. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

**G. Lenzini**. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

**I. Kurtev**. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

**T. Wolle**. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

**O. Tveretina**. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

**A.M.L. Liekens**. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

**J. Eggermont**. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

**B.J. Heeren**. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

**G.F. Frehse**. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

**M.R. Mousavi**. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15