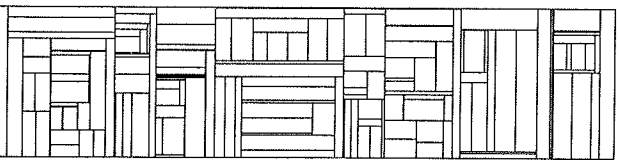
ISSN 0105-8517

An Introduction to Event Structures

Glynn Winskel

DAIMI PB – 278
April 1989

AARHUS UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
Ny Munkegade 116 – DK 8000 Aarhus C – DENMARK
Telephone: + 45 6 12 71 88 *Telex: 64767 aausci dk*



An introduction to event structures

by
Glynn Winskel

Computer Science Department,
Aarhus University,
Denmark.

ABSTRACT: Event structures are models of processes as events constrained by relations of consistency and enabling. These notes are intended to introduce the mathematical theory of event structures, show how they are related to Petri nets and Scott domains, and how they can be used to provide semantics to programming languages for parallel processes as well as languages with higher types.

Key words: Event structures, Petri nets, traces, concurrency, nondeterminism, parallel computation, semantics, communicating processes, higher types, lambda calculus.

CONTENTS

0. Introduction.

Event structures are models of processes as events constrained by relations of consistency and enabling. Their study in denotational semantics first arose as a biproduct in the pioneering work of G.Kahn and G.Plotkin on some foundational questions in denotational semantics (see [KP]). The concrete data structures of Kahn and Plotkin were later realised to be closely related to confusion-free Petri nets (see [NPW]) and this led to the more general definitions discussed here. Since then they have been developed as a model in their own right and for certain applications (*e.g.* see section 7 on higher-type event structures) they are easier and less clumsy to use than Petri nets, to which they are closely related however. These notes are intended to introduce the mathematical theory of event structures, show how they are related to Petri nets and Scott domains, and how they can be used to provide semantics to programming languages for parallel processes as well as languages with higher types.

The notes [W1] provide another description of event structures, in many ways fuller than the presentation here. They overlap a great deal with the notes here, and fairly often the reader is referred to [W1] for proofs or further details. These notes do however try to compensate for the terse presentation in [W1] and should be easier to read.

1. Modelling concurrency.

The models of computation we shall consider in these notes are based on the primitive notion of an *event*. We all have an intuitive idea, from everyday experience and science, of what an event is. Attempting a rough definition we might say an event is an action which one can choose to regard as indivisible—it either has happened or has not according to our description of some process. This is not to say that an event is indivisible, and without detailed structure, in any absolute sense; it might well have internal structure, and consist of a complicated process, which it is sensible to analyse at another level of abstraction. But then, of course, at that more detailed level of abstraction what was originally an event is no longer a single event, but several or many. ¿From their far perspective, historians may talk of the event of a battle or the birth of a famous person—not just single events to the people involved at the time! An event can have detailed structure in another sense—its occurrence may be very significant a change a great deal—though this is determined more by how the event influences other events. How we catch this will be discussed

shortly. Another property we expect of an event is that it is localised in space and time, that as far as our description is concerned it occurs in a small area and over a small period of time. Speaking informally, this really follows from our understanding of an event as being without detailed structure—if we were to understand an event as occupying some extended region of space and time then its dimensions would presumably be important. Again, of course, there is nothing absolute about this; what we think of as small depends on what we are modelling and how we go about it.

In viewing the events of a distributed computation, it may well be that we can ascribe precise places and times to all the events of interest. True, if the computation is very distributed, so that relativistic effects become important, these may not be agreed on by all observers. But even without relativistic effects, and even if it is feasible, there is generally no point in analysing the computation at such a level of detail—the precise places and times are most often incidental details. What is important in designing and analysing distributed computations are the significant events and how the occurrence of an event causally depends on the previous occurrence of others. For example, the event of a process transmitting a message would presumably depend on it first performing some events, so it was in the right state to transmit, including the receipt of the message which in turn would depend on its previous transmission by another process. This outlook has been proposed by Lamport among others (see *e.g.* [Lam]).

The scale at which it is sensible to view a computation as distributed can vary immensely. For example, similar ideas have been used in the analysis of self-timed circuits in VLSI (See *e.g.* [Rem]).

Such ideas suggest that we view distributed computations as event occurrences together with a relation expressing causal dependency, and this we may reasonably take to be a partial order. As a definition we take:

1.1 Definition. An *elementary event structure* $(E, \leq)$ is a partially ordered set. The set E is to be thought of as a set of event occurrences and the partial order relation as expressing *causal dependency*; for two events e, e' we have $e \leq e'$ when the occurrence of the event e' depends on the previous occurrence of the event e.

Guided by our interpretation we can formulate a notion of computation state of an elementary event structure $(E, \leq)$. Taking a computation state of a process to be represented by the set x of events which have occurred in the computation, we expect that

$$e' \in x \ \ \& \ \ e \leq e' \Rightarrow e \in x;$$

if an event has occurred then all events on which it causally depends have occurred too. Say a subset $x \subseteq E$ which satisfies this property is *left-closed*, and collect all such subsets together in the family described as $\mathcal{L}(E)$.

A particular left-closed set is determined by an event e of an event structure $(E, \leq)$. Define

$$\lceil e \rceil = \{e' \in E \mid e' \leq e\},$$

which is clearly left-closed.

Viewing computation states as such subsets, progress in a computation is measured by the occurrence of more events. Let $x, y \in \mathcal{L}(E)$ for an event structure E. If $x \subseteq y$ then x can be regarded as a subbehaviour of y. The relation of inclusion between left-closed subsets is an information order of the sort familiar from denotational semantics, but special in that more information corresponds to more events having occurred. The least element of information in an order $(\mathcal{L}(E), \subseteq)$ is the empty set, when no events have occurred, and there is a maximum element, the set containing all events. In fact the partial orders of left-closed subsets are complete lattices. Recall:

1.2 Definition. A complete lattice is a partial order which has least upper bounds (joins or suprema) $\bigsqcup X$ and greatest lower bounds (meets or infima) $\prod X$ of arbitrary subsets X. We write $x \sqcup y$ and $x \sqcap y$ for the least upper bound and greatest lower bound respectively of two elements x, y.

As we shall see the lattices associated with left-closed sets are lattices of a rather special sort.

This view of a process as a domain of computation states is a little non-standard and unusual—one is used to processes being denotated by elements of a domain, not by a domain itself. Domains are more usually used as denotations of types. However, thinking of a domain of computation states $\mathcal{L}(E)$ as the "type" of computation states of a process makes the idea less strange, and more familiar.

One can ask: precisely what class of domains are represented in this way? In fact the class is exactly that of algebraic lattices which are infinitely distributive in that they satisfy the following laws:

$$(\bigsqcup X) \sqcap y = \bigsqcup \{x \sqcap y \mid x \in X\} \tag{1}$$

$$(\prod X) \sqcup y = \prod \{x \sqcup y \mid x \in X\} \tag{2}$$

It is straightforward to verify one part of this statement. First, recall the definition of algebraic lattice.

1.3 Definition.

A *directed subset* of a partial order L is a subset $S \subseteq L$ with the property that for any finite set $X \subseteq S$ there is an element $s \in S$ such that $\forall x \in X.\ x \sqsubseteq s$.

(In particular, a chain is directed.)

A *finite element* of a complete lattice is an element f with the property that for all directed sets S, if $f \sqsubseteq \bigsqcup S$ then there is some $s \in S$ for which $f \sqsubseteq s$.

A complete lattice is *algebraic* if for any element d the set $\{x \sqsubseteq d \mid x \text{ is finite}\}$ is directed and has least upper bound d.

1.4 Theorem. *Let $(E, \leq)$ be a partial order. Then $(\mathcal{L}(E), \subseteq)$ is an algebraic lattice which satisfies the distributive laws (1) and (2) above.*

Proof. Verifying that left-closed subsets of a partial order, ordered by inclusion, form an algebraic lattice with meets and joins given as intersections and unions is routine, as is the verification of the distributivity laws. ∎

The converse is harder. A first, simpler representation of the lattices represented by elementary event structures starts by observing that an event e in an event structure corresponds with the left-closed set

$$\lceil e \rceil = \{e' \mid e' \leq e\}.$$

Such configurations are characterised in the domain order as being complete primes. Moreover as every left-closed subset is the union of such configurations they form a subbasis in the domain of configurations of an event structure.

1.5 Definition. Let $(L, \sqsubseteq)$ be a complete lattice.

A *complete prime* of Ł is an element $p \in L$ such that

$$p \sqsubseteq \bigsqcup X \Rightarrow \exists x \in X.\ p \sqsubseteq x$$

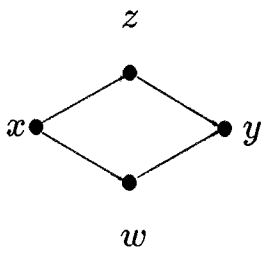
for any set X.

Ł is *prime algebraic* iff

$$x = \bigsqcup \{p \sqsubseteq x \mid p \text{ is a complete prime}\},$$

for all $x \in L$.

1.6 Example. Consider the lattice:



As in any finite lattice, all the elements are finite, and the lattice is algebraic. The least element w, like all least elements, is the least upper bound $\bigsqcup\emptyset$ without there being an element in $\emptyset$ which dominates it. It can not therefore be a complete prime. The element z is dominated, in fact equal to, the least upper bound $x \bigsqcup y$ without being dominated by either x or y. It can not be a complete prime either. On the other hand both x and y are complete primes—any least uper bound of a set dominating them must contain an element which does so.

1.7 Proposition. *Let E be an elementary event structure. In the partial order $(\mathcal{L}(E), \subseteq)$ the complete primes are precisely those left-closed subsets of the form $\lceil e \rceil$ for $e \in E$.*

Conversely, any complete lattice which is prime algebraic domain is associated with an elementary event structure in which the events are its complete primes.

1.8 Definition. Let L be a complete lattice which is prime algebraic. Define $\mathcal{P}r(L) = (P, \leq)$, where P consists of the complete primes of L and

$$p \leq p' \Leftrightarrow p \sqsubseteq p'$$

for $p, p' \in P$

1.9 Theorem. *Let L be a complete lattice which is prime algebraic. Then $\mathcal{P}r(L)$ is an elementary event structure, with $\phi : L \cong (\mathcal{LP}r(L), \subseteq)$ giving an isomorphism of partial orders where*
$\phi(d) = \{p \sqsubseteq d \mid p \text{ is a complete prime}\}$ with inverse $\theta : \mathcal{LP}r(L) \to L$ given by $\theta(x) = \bigsqcup x$.

Proof. Let P be the complete primes of L. Obviously the maps θ and ϕ are monotonic *i.e.* order preserving. We show they are mutual inverses and so give the required isomorphism.

Firstly we show $\theta\ \phi = 1$. Thus we require $x = \bigsqcup\{p \in P \mid p \sqsubseteq x\}$ for all $x \in L$. But this is just the condition of prime algebraicity.

Now we show $\phi\ \theta = 1$. Let $X \in \mathcal{L}(P, \leq)$. We require $X = \phi\ \theta(X)$ *i.e.* $X = \{p \in P \mid p \sqsubseteq \bigsqcup X\}$. Clearly $X \subseteq \{p \in P \mid p \sqsubseteq \bigsqcup X\}$. Conversely if $p \sqsubseteq \bigsqcup X$, where p is a complete prime, then certainly $p \sqsubseteq q$ for some $q \in X$. However X is left–closed so $p \in X$, showing the converse inclusion.

Thus we have established the required isomorphism. ∎

To show prime algebraicity for infinitely distributive, algebraic lattices we use another idea. Events in $(E, \leq)$ also manifest themselves in the lattice $(\mathcal{L}(E), \subseteq)$ as prime intervals. We say x is *covered* by x' in a partial order, written $x \prec x'$ iff

$$x \sqsubseteq x' \ \ \& \ \ x \neq x' \ \ \& \ \ (\forall z.\ x \sqsubseteq z \sqsubseteq x' \Rightarrow x = z \ \text{ or } \ z = x').$$

The relation $\prec$ is called the *covering* relation. A *prime interval* is a pair $[x, x']$ such that $x \prec x'$. In $(\mathcal{L}(E), \subseteq)$ a prime interval is associated with the occurrence of an event at some element $x \in \mathcal{L}(E)$; in $(\mathcal{L}(E), \subseteq)$, the relation $x \prec x'$ holds iff there is an event e such that $e \notin x$ and $x' = x \cup \{e\}$ with $x, x' \in \mathcal{L}(E)$. Note that an event is not in general associated with a unique prime interval but many.

In a complete lattice which is prime algebraic, $x \prec x'$ means there is a unique complete prime p such that $x' = x \sqcup p$, and in fact p can be recovered as

$$p = \prod\{z \mid x' \sqsubseteq x \sqcup z\}.$$

This observation and the following lemma, which ensures there are enough prime intervals, give the heart of the proof.

1.10 Lemma. *Let $Ł = (L, \sqsubseteq)$ be an algebraic lattice. Then*

$$\forall x, y \in Ł.\ x \sqsubseteq y \ \ \& \ \ x \neq y \Rightarrow \exists z, z' \in Ł.\ x \sqsubseteq z \prec z' \sqsubseteq y.$$

Proof. Suppose x, y are distinct elements of Ł such that $x \sqsubseteq y$. Because Ł is algebraic there is a finite element b such that $b \not\sqsubseteq x \ \ \& \ \ b \sqsubseteq y$. By Zorn's lemma there is a maximal chain C of elements above x and strictly below $x \sqcup b$. As b is finite, from the construction of C we must have $x \sqsubseteq \bigsqcup C \prec x \sqcup b \sqsubseteq y$. ∎

1.11 Theorem. *Let Ł be a complete lattice. Then Ł is prime algebraic iff it is algebraic and infinitely distributive (i.e. satisfies the distributive laws (1) and (2)).*

Proof.

"*only if*":

Let Ł be a prime algebraic complete lattice. Let P be the ordering of Ł restricted to its complete primes. By the previous theorem we know Ł $\cong \mathcal{L}(\mathrm{P})$ so it is sufficient to prove properties for $\mathcal{L}(\mathrm{P})$. We have already seen the distributivity laws follow from the corresponding laws for sets.

The finite elements of $(\mathcal{L}(\mathrm{P}), \subseteq)$ are easily shown to be precisely the left-closures of finite subsets of P. Suppose $x \in \mathcal{L}(\mathrm{P})$ is finite. Obviously $x = \bigcup\{[X] \mid X \subseteq_{fin} x\}$. But the set $\{[X] \mid X \subseteq_{fin} x\}$ is clearly directed so, because x is finite, $x = [X]$ for some finite set $X \subseteq P$. Conversely, it is clear that an element of the form $[X]$, for a finite $X \subseteq P$, is necessarily finite; if $[X] \subseteq \bigcup S$ for a directed subset S of $\mathcal{L}(\mathrm{P})$ then X, and so $[X]$, is included in the union of a finite subset of S, and so in an element of S. Clearly now every element of $\mathcal{L}(\mathrm{P})$ is the least upper bound of the finite elements below it, making $\mathcal{L}(\mathrm{P})$ algebraic.

Thus Ł is an algebraic lattice satisfying the distributive laws (1) and (2).

"*if*":

Let Ł $= (L, \sqsubseteq)$ be an algebraic lattice satisfying the distributive laws (1) and (2).

Let $x \prec x'$ in Ł. Define $pr[x, x'] = \bigsqcap\{y \in L \mid x' \leq x \sqcup y\}$. We show $p = pr[x, x']$ is a complete prime of Ł. Note first that $x \sqcup p = \bigsqcap\{x \sqcup y \mid x' \sqsubseteq x \sqcup y\} = x'$ by distributive law (2). Now suppose $p \sqsubseteq \bigsqcup Z$ for some $Z \subseteq L$. Then $p = (\bigsqcup Z) \sqcap p = \bigsqcup\{z \sqcap p \mid z \in Z\}$ by the distributive law (1). Write $Z' = \{z \sqcap p \mid z \in Z\}$, so $p = \bigsqcup Z'$. Then $x' = x \sqcup p = x \sqcup (\bigsqcup Z') = \bigsqcup\{x \sqcup z' \mid z' \in Z'\}$. Clearly $x \sqsubseteq x \sqcup z' \sqsubseteq x'$ for all $z \in Z'$. As $x \prec x'$ we must have $x' = x \sqcup z'$ for some $z' \in Z'$; otherwise $x = x \sqcup z'$ for all $z' \in Z'$ giving the contradiction $x = \bigsqcup\{x \sqcup z' \mid z' \in Z'\} = x'$. But then $p \sqsubseteq z'$ from the definition of p. However $z' = z \sqcap p$ for some $z \in Z$. Therefore $p \sqsubseteq z$ for some $z \in Z$. Thus p is a complete prime of Ł.

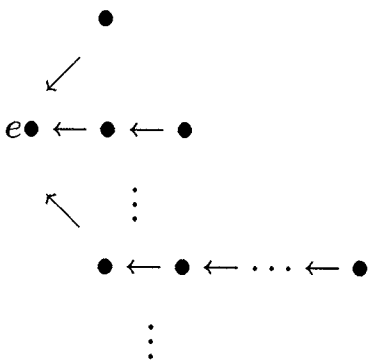
That Ł is prime algebraic follows provided for $z \in L$, we have $z = \bigsqcup\{pr[x, x'] \mid x \prec x' \sqsubseteq z\}$. Let $z \in L$. Write $w = \bigsqcup\{pr[x, x'] \mid x \prec x' \sqsubseteq z\}$. Clearly $w \sqsubseteq z$. Suppose $w \neq z$. Then, by the lemma, $w \sqsubseteq x \prec x' \sqsubseteq z$ for some $x, x' \in L$. Write $p = pr[x, x']$. Then $p \sqsubseteq w$ making $x \sqcup p = x$, a contradiction as $x \sqcup p = x'$. Thus each element of Ł is the least upper bound of the complete primes below it, as required.

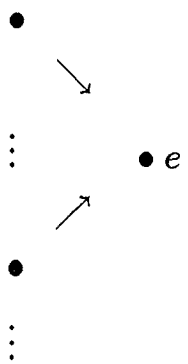
Thus we have established the required equivalence between prime algebraic complete lattices and algebraic lattices satisfying (1) and (2). ∎

So far we have fed very little intuition into our definition of event structure. True, our interpretation of the partial order as one of causal dependency motivated our choice of formulation of computation state. But our physical understanding of what events are invokes more than is caught there. For example, should it be possible for an event to occur when it causally depends on an infinite set of events, as in the following examples? In the diagrams we represent a single link in the causal dependency relation, say $e_0 < e_1$, by drawing $e_0\bullet \rightarrow \bullet e_1$ or $e_1\bullet \leftarrow \bullet e_0$.

$$e_0\bullet \rightarrow e_1\bullet \rightarrow e_2\bullet \rightarrow \cdots e_n\bullet \rightarrow \cdots \bullet e$$

$$\cdots \rightarrow e_n\bullet \rightarrow \cdots \rightarrow e_2\bullet \rightarrow e_1\bullet \rightarrow e\bullet$$





In the first example, an event e can only occur after a chain of events, first e_0, then e_1, then e_2 *etc.* , have occurred. Such an event structure is of the kind that arises in describing the processes in the paradoxes of Zeno. Zeno, to point

out the illusory nature of reality, argued that a door could never close because the event of it doing so would depend on the events of it first half closing, then three-quarters closing, then seven-eighths closing *etc.* . Nowadays, familiar as we are with the calculus and the physics of continuous processes, it is hard to appreciate the difficulties Zeno saw. But still, we would not expect such an event structure to arise when describing processes where the events are discrete, meaning roughly that the events do not blur into one another. Attempting a tentative definition of what we mean by "discrete", we shall say a set of events in space and time is discrete when we can uniformly choose some real number r so that any two events can be separated by spheres of space and time of radius r.*

The remaining examples are a little more subtle. The second example represents a process where in order for an event e to occur e_1 must have occurred before, before that e_2 *etc.* . The third example represents a process where the occurrence of an event e depends on the previous occurrence of chains of events of unbounded length. The fourth shows an event e whose occurrence depends on the previous occurrence of an infinite number of events all independent of each other. When the events are understood to be discrete the reasonableness of these three descriptions is determined by whether or not there is an initial state, at which no events have occurred. If there is such an initial state the second, third and fourth examples cannot be discrete.

To see this, we use the fact there is an upper bound on the speed at which causal influence can travel. Any point in space and time determines a future-cone of points in space and time which it can effect and a past-cone of points which can have affected it. Assume a process begins with some "initial event" i, marking a state where no process events have occurred, and includes the occurrence of an event e. Then the intersection of the future-cone of i and the past-cone of e is a bounded closed region which therefore has the property that any infinite subset of points has an accumulation point. Because this region must contain all the process events on which e depends we cannot have discreteness for any of the examples above.

In fact the argument entails that to obtain discreteness under the assumption that there is an initial state, at which no events have occurred, we

* Another definition which works equally well for our purposes is to say a set of events in space and time is discrete when it is a closed subset such that any two events can be separated by an open neighbourhood. I do not know how to choose between the two definitions.

should insist on the following strong discreteness axiom on event structures.

1.12 Definition. Say an elementary event structure satisfies the *axiom of finite causes* when

$$\forall e \in E.\ \{e' \in E \mid e' \leq e\} \text{ is finite.}$$

We have just seen an attempt to argue for an axiom on event structures based on simple physical principles. As a computational argument for the axiom we shall show how it is implied by Scott's thesis, once we make certain assumptions about how to get datatypes and functions between them from an elementary event structure. Dana Scott proposed the thesis that computable functions between datatypes are continuous, it being understood that datatypes are associated with domains of information and that computable functions between datatypes are associated with functions between their domains of information. A function $f : D \to E$ from one complete lattice D to another E is *continuous* iff it preserves least upper bounds of directed sets *i.e.* for all directed sets S

$$\bigsqcup fS = f(\bigsqcup S).$$

Note a continuous function is *monotonic*, *i.e.*

$$\forall x, y \in D.\ x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y),$$

In particular, a continuous function should preserve least upper bounds of ω-chains, *i.e.* for all chains $x_0 \sqsubseteq x_1 \sqsubseteq \cdots \sqsubseteq x_n \sqsubseteq \cdots$ in D we have

$$\bigsqcup_{n\in\omega} f(x_n) = f(\bigsqcup_{n\in\omega} x_n).$$

Intuitively the ultimate output value should be no more than the limit of the values determined at finite stages in delivering the input, so we can approximate the ultimate output value arbitrarily closely by the output values at finite stages. Scott's thesis has an intuitive justification (see *e.g.* [St]), and plays a key part in the mathematical basis of denotational semantics.

We show $\mathrm{E} = (E, \leq)$ will obey Scott's thesis iff it satisfies the axiom of finite causes. Of course we need to make clear what we mean by "obey Scott's thesis". This hinges on associating datatypes and continuous functions with E.

We can choose to imagine some of the events of E as being events of input E_0 from some datatype, some as internal events, and others as events of output E_1 to some datatype. The datatypes may have their own causal dependencies, which contribute to the dependency of the full process, so the input datatype can carry an partial order $\mathrm{E}_0 = (E_0, \leq_0)$ and the output datatype a partial order $\mathrm{E}_1 = (E_1, \leq_1)$. The orderings of the datatypes should be sub-partial orders of that of the process, *i.e.*

$$\mathrm{E}_0 \subseteq \mathrm{E} \;\&\; \mathrm{E}_1 \subseteq \mathrm{E},$$

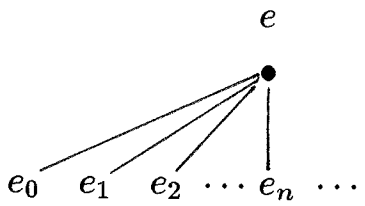
meaning $\leq_0 \subseteq \leq$ and $\leq_1 \subseteq \leq$. There are natural domains of information associated with the two datatypes, *viz.* their domains of left-closed sets of events. The process induces a function between the domains. Define

$$f_{E_0,E_1} : \mathcal{L}(\mathrm{E}_0) \to \mathcal{L}(\mathrm{E}_1) \text{ to map } x \mapsto \{e \in E_1 \mid \lceil e \rceil \cap E_0 \subseteq x\}.$$

The idea is that an event of E occurs once the necessary input events have occurred. It is clear that:

1.13 Proposition. *The function f_{E_0,E_1} is monotonic.*

However for partial orders in general the function may not be continuous. Consider, for example, the partial order



with $E_0 = \{e_n \mid n \in \omega\}$ and $E_1 = \{e\}$ ordered by the identity relation. Then taking S to be the directed set consisting of all finite subsets of E_0 we see (as in the proof of the theorem below) that the least upper bound of S is not preserved by f_{E_0,E_1}. If E is to represent a computable process, according to Scott's thesis, f_{E_0,E_1} should be continuous. Furthermore it should be for any choice of events for the input and output datatypes.

1.14 Definition. We say E *obeys Scott's thesis* iff

$$\forall \mathrm{E}_0, \mathrm{E}_1.\ (\mathrm{E}_0 \subseteq \mathrm{E} \;\&\; \mathrm{E}_1 \subseteq \mathrm{E} \Rightarrow f_{E_0,E_1} \text{ is continuous}).$$

Now by a simple argument we can show those elementary event structures E which obey Scott's thesis are precisely those which satisfy the axiom of finite causes.

1.15 Theorem. *The elementary event structure E obeys Scott's thesis iff it satisfies the axiom of finite causes.*

Proof.

"only if" Suppose E obeys Scott's thesis. Suppose for some e in E we had $\lceil e \rceil$ infinite. Take

$$E_0 = \{e' \in E \mid e' < e\} \text{ and } E_1 = \{e\},$$

with both ordered by the identity relation. Define S to consist of all finite subsets of E_0. Then S is a directed subset of $\mathcal{L}(\mathrm{E}_0)$. Moreover no element of S is E_0 as E_0 is infinite. However now $f_{E_0,E_1}(\bigcup S) = \{e\}$ while $\bigcup f_{E_0,E_1} S = \emptyset$. Thus f_{E_0,E_1} is not continuous which contradicts the assumption that E obeys Scott's thesis. Thus $\lceil e \rceil$ is finite for all $e \in E$.

"if" Suppose $\lceil e \rceil$ is finite for all e in E. Assume $E_0 \subseteq E$ and $E_1 \subseteq E$. Let S be a directed subset of $\mathcal{L}(\mathrm{E}_0)$. Abbreviate f_{E_0,E_1} to f. As f is always monotonic we have $\bigcup fS \subseteq f(\bigcup S)$. Suppose $e \in f(\bigcup S)$. Then $\lceil e \rceil \cap E_0 \subseteq \bigcup S$. As $\lceil e \rceil$ is finite so is $\lceil e \rceil \cap E_0$. Thus because S is directed $\lceil e \rceil \cap E_0 \subseteq s$ for some $s \in S$. Then $e \in f(s)$. This shows $f(\bigcup S) \subseteq \bigcup fS$ so $f(\bigcup S) = \bigcup fS$. Therefore f is continuous. Hence $(E, \leq)$ obeys Scott's thesis, as required. ∎

It should be stressed that the argument does rest on assumptions about how to obtain datatypes and functions between them from an elementary event structure; in the presence of, for example, a topology on events, when they are not discrete, these are likely to be obtained differently.

It is a slippery business arguing for axioms on event structures, resting as it does on assumptions about common understanding and intuitions. Henceforth we shall just assume the axiom of finite causes, or an equivalent, for our models of processes.

The axiom of finite causes on event structures has its lattice-theoretic analogue.

1.16 Definition. Say an algebraic lattice is *finitary* iff every finite element dominates only a finite number of elements, *i.e.* $\{d \mid d \sqsubseteq x\}$ is finite for every finite element x.

Not surprisingly, the algebraic lattices represented by elementary event structures satisfying the axiom of finite causes are finitary. Because of this

the infinite distributivity laws are implied by distributivity, just in a finite form.

1.17 Definition. Say a lattice is *distributive* iff it satisfies

$$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z) \qquad (3).$$

Theorem. *Let Ł be an an algebraic lattice which is finitary. Then Ł is prime algebraic iff Ł satisfies the finite distributive law (3).*

1.18 Corollary.
(i) Let E be an elementary event structure satisfying the axiom of finite causes. Then $(\mathcal{L}(E), \subseteq)$ is an algebraic lattice which is finitary and satisfies the finite distributive law (3).
(ii) Let Ł be an algebraic lattice which is finitary and satisfies the finite distributive law (3). Then there is an elementary event structure E satisfying the axiom of finite causes such that $Ł \cong (\mathcal{L}(E), \subseteq)$.

A word on our formulation of computation state as a left-closed subset: Here the most generous formulation has been given, in that it allows the most general set of left-closed subsets. It includes all infinite left-closed subsets of events which according to our interpretation could only be realised after an infinite period of time. A more refined analysis of what is to be meant by "computation state" might rule out some of them. Questions of "fairness" might rule out all infinite left-closed subsets but the maximal one.

2. Adding nondeterminism.

So far we have postulated that concurrency or parallelism is to be modelled through a partial order expressing causal dependency. Clearly, one thing is missing from such descriptions, and this is the ability to model a process which, perhaps influenced by the environment, can behave in different and incompatible ways, the phenomenon of nondeterminism. To model nondeterminism we adjoin further structure in the form of a conflict relation to express how the occurrence of certain events rules out the occurrence of others. In these notes we shall assume that events exclude each other in a binary fashion (see [W1] for a more general treatment). As an example of such a binary conflict, the event of an "a" being typed in the first position of a line conflicts with the event of a "b" being typed in that same position.

2.1 Definition. Define a *prime event structure* to be a structure $\mathrm{E} = (E, \#, \leq)$ consisting of a set E, of *events* which are partially ordered by $\leq$, the *causal dependency relation*, and a binary, symmetric, irreflexive relation $\# \subseteq E \times E$, the *conflict relation*, which satisfy

$$\{e' \mid e' \leq e\} \text{ is finite,}$$
$$e\#e' \leq e'' \Rightarrow e\#e''$$

for all $e, e', e'' \in E$.

2.2 Definition. Let $(E, \#, \leq)$ be a prime event structure. Define its *configurations*, $\mathcal{L}(\mathrm{E})$, to consist of those subsets $x \subseteq E$ which are

conflict-free: $\forall e, e' \in x.\ \neg(e\#e')$ and

left-closed: $\forall e, e'.\ e' \leq e \in x \Rightarrow e' \in x$.

In particular, define $\lceil e \rceil = \{e' \in E \mid e' \leq e\}$.

We study the kind of domains which are represented by prime event structures. As we have chosen to work with a binary conflict relation we rightly expect that compatibility in the domain of configurations will be determined in a binary fashion.

2.3 Definition. Let $(D, \sqsubseteq)$ be a partial order.

Say a set $X \subseteq D$ is *finitely compatible* iff if every finite $Y \subseteq X$ has an upper bound.

(A directed set is finitely compatible.)

Say D is *consistently complete* iff every finitely compatible subset $X \subseteq D$ has a least upper bound $\bigsqcup X$.

A partial order is a *Scott domain* (or simply a domain) iff it is consistently complete and the restriction of the order to $\{x \in D \mid x \sqsubseteq d\}$ is an algebraic lattice for all $d \in D$; its *finite elements* are precisely those elements which are finite in some such lattice.

Say D is *coherent* iff all subsets $X \subseteq D$ such that

$$\forall d_0, d_1 \in X.\ d_0 \uparrow d_1$$

have least upper bounds $\bigsqcup X$.

(Note a consistent complete partial order has a least element, *viz.* $\bot = \bigsqcup \emptyset$, though it may not have a greatest. It follows the same is true also for coherent partial orders.)

Remark. Note any consistent complete partial order has greatest lower bounds of any nonempty subsets: For X nonempty, $\prod X = \bigsqcup \{d \mid \forall x \in X.\ d \sqsubseteq x\}$.

There is another characterisation of finite elements of a domain:

2.4 Proposition. *Let $(D, \sqsubseteq)$ be a domain, as defined above. An element $x \in D$ is finite iff for every directed subset S of D $x \sqsubseteq \bigsqcup S$ implies there is some $s \in S$ for which $x \sqsubseteq s$.*

Proof. The proof of the proposition uses the fact that meets in an algebraic lattice L are continuous *i.e.*

$$\bigsqcup S \sqcap d = \bigsqcup \{s \sqcap d \mid s \in S\}$$

for all elements d and directed subsets S. It is clear that $\bigsqcup \{s \sqcap d \mid s \in S\} \sqsubseteq \bigsqcup S \sqcap d$. To see the converse, let x be a finite element of L with $x \sqsubseteq \bigsqcup S \sqcap d$. Then as $x \sqsubseteq \bigsqcup S$, a directed set, there is an $s \in S$ so $x \sqsubseteq s$. Hence $x \sqsubseteq s \sqcap d$, giving $x \sqsubseteq \bigsqcup \{s \sqcap d \mid s \in S\}$. In an algebraic lattice this is enough to establish the converse ordering, and hence the equality.

Suppose x is an element of D with the property that for every directed subset S if $x \sqsubseteq \bigsqcup S$ then $x \sqsubseteq s$ for some $s \in S$. Take any d such that $x \sqsubseteq d$. The element x is finite in the lattice $\{y \in D \mid y \sqsubseteq d\}$, and so is finite in D. Conversely, suppose x is finite in D. Then x is finite in a lattice $L = \{y \in D \mid y \sqsubseteq d\}$ for some $d \in D$. Suppose S is a directed subset of D such that $x \sqsubseteq \bigsqcup S$. Then by the fact above

$$x \sqsubseteq \bigsqcup S \sqcap d = \bigsqcup \{s \sqcap d \mid s \in S\},$$

the least upper bound of a directed set in the lattice L. Thus $x \sqsubseteq s \sqcap d$, so $x \sqsubseteq s$, as required. ∎

For those familiar with another definition of domain, we remark that the proposition above is the key to showing our definition of domain is equivalent to the usual definition.

As in the section on elementary event structures, the simplest representation of the domains represented by prime event structures starts by observing that an event e in an event structure corresponds with the configuration $\lceil e \rceil$. Such elements are characterised as being complete primes, extending the definition used before.

2.5 Definition.

A domain D is a *prime algebraic* iff each sublattice $\{x \in D \mid x \sqsubseteq d\}$ is prime algebraic; the complete primes of D are those elements which are complete primes in any sublattice $\{x \in D \mid x \sqsubseteq d\}$.

A domain D is a *distributive (respectively infinitely)* iff each sublattice $\{x \in D \mid x \sqsubseteq d\}$ is distributive (respectively infinitely).

A domain D is a *finitary* iff each sublattice $\{x \in D \mid x \sqsubseteq d\}$ is finitary.

¿From earlier results it follows that several concepts coincide. For example it is a direct consequence of the definition and the result 1.11 that a domain is prime algebraic iff it is infinitely distributive. Similarly, if a domain is finitary then it is prime algebraic iff it is distributive. Elsewhere, in [NPW,W,W1], other definitions have been given of the concepts above so we spend a moment checking the new definitions agree with the old. A standard way to say a domain is distributive is expressed in the next proposition as an equivalent to the definition above.

2.6 Proposition. *A domain is distributive iff it satisfies:*

$$x \uparrow y \Rightarrow (x \sqcup y) \sqcap z = (x \sqcap z) \sqcup (y \sqcap z) \qquad (*)$$

Proof. Suppose the condition $(*)$ holds in some domain D. Then certainly any sublattice $\{x \in D \mid x \sqsubseteq d\}$ is distributive for any $d \in D$. Conversely, suppose D is distributive in the sense of the definition above. Let $x, y, z \in D$ with $x \uparrow y$. Then $x, y \sqsubseteq d$ for some $d \in D$. Take $z' = z \sqcap d$. Then, as the sublattice $\{x \in D \mid x \sqsubseteq d\}$ is distributive with elements x, y, z'

$$(x \sqcup y) \sqcap z = (x \sqcup y) \sqcap z' = (x \sqcap z') \sqcup (y \sqcap z') = (x \sqcap z) \sqcup (y \sqcap z).$$

∎

Similar, more standard, reformulations are possible for the definitions of the stronger notions of distributivity of a domain.

2.7 Proposition. *Let D be a prime algebraic domain. Its complete primes are precisely those elements p with the property that for any compatible subset $X \subseteq D$ if $p \sqsubseteq \bigsqcup X$ then $p \sqsubseteq x$ for some $x \in X$.*

We remark that the fact above is the key to showing the present definition and the alternative definition of prime algebraicity in [W,NPW,W1] are equivalent.

Earlier results make it straightforward to characterise the order of configurations of prime event structures.

2.8 Theorem. *Let E be a prime event structure. The partial order $(\mathcal{L}(E), \subseteq)$ is a coherent, finitary prime algebraic domain; the complete primes are the set $\{\lceil e \rceil \mid e \in E\}$.*

Conversely, any coherent, finitary prime algebraic domain is associated with a prime event structure in which the events are its complete primes.

2.9 Definition. Let D be a coherent, finitary prime algebraic domain. Define $\mathcal{P}r(\mathrm{D}) = (P, \#, \leq)$, where P consists of the complete primes of D,

$$p \leq p' \Leftrightarrow p \sqsubseteq p',$$

and

$$p \# p' \Leftrightarrow p \not\uparrow p',$$

for $p, p' \in P$

2.10 Theorem. *Let D be a coherent, finitary prime algebraic domain. Then $\mathcal{P}r(D)$ is a prime event structure, with $\phi : D \cong (\mathcal{L}\mathcal{P}r(D), \subseteq)$ giving an isomorphism of partial orders where*
$\phi(d) = \{p \sqsubseteq d \mid p$ is a complete prime$\}$ with inverse $\theta : \mathcal{L}\mathcal{P}r(D) \to D$ given by $\theta(x) = \bigsqcup x$.

Thus prime event structures and coherent, finitary prime algebraic domains are equivalent; one can be used to represent the other.

As we have seen in section 1, events also manifest themselves in a domain of configurations as prime intervals. Recall a *prime interval* is a pair $[d, d']$ such that $d \prec d'$. In a domain of configurations a prime interval is associated with the occurrence of an event at some configuration; in a domain of configurations $(\mathcal{L}(E), \subseteq)$, the relation $x \prec x'$ holds iff there is an event e such that $e \notin x$ and $x' = x \cup \{e\}$ with $x, x' \in \mathcal{L}(E)$. Define

$$[c, c'] \leq [d, d'] \Leftrightarrow d' = c' \sqcup d \ \ \& \ \ c = c' \sqcap d.$$

Form the equivalence relation $\sim$ as the symmetric, transitive closure of $\leq$, and write $[d, d']_\sim$ for the equivalence class of $[d, d']$ with respect to $\sim$. In a domain of configurations, $[c, c'] \sim [d, d']$ implies $c' \setminus c = d' \setminus d = \{e\}$ for the same event e. So $\sim$-classes are associated with unique events. For domains represented as families of configurations of complete primes this association is a 1-1 correspondence.

2.11 Proposition. *Let D be a coherent, finitary prime algebraic domain. Let $\phi : D \cong \mathcal{L}\mathcal{P}r(D)$ be the isomorphism $d \mapsto \{p \sqsubseteq d \mid p$ is a complete prime$\}$. Define the following map from $\sim$-classes to complete primes:*

$$[d, d']_\sim \mapsto p$$

where p is the unique member of $\phi(d')\backslash\phi(d)$. This map is a 1-1 correspondence with inverse

$$p \mapsto [d, d']_\sim$$

where $d = \bigsqcup\{c \mid c \sqsubseteq p \ \ \& \ \ c \neq p\}$ *and* $d' = p$.

Sometimes one makes use of the fact that if d is a finite element of a coherent, finitary prime algebraic domain D then there is a *covering chain*

$$\bot = d_0 \prec d_1 \prec \cdots \prec d_n = d$$

in D up to d. This is obvious because we can represent any such domain as the left closed consistent subsets of some prime event structure.

Another characterisation of finitary, coherent prime algebraic domains can be obtained from the results in the last section.

2.12 Theorem. *A finitary, coherent domain is prime algebraic iff it is distributive.*

Remark. Prime event structures represent finitary, coherent prime algebraic domains. These are precisely the coherent dI-domains of Gérard Berry (see [Be]).

3. Stable event structures.

Not all the constructions we want to use are easily defined on prime event structures. For example, parallel compositions and function space constructions are not easily defined directly on them and to do so involves fairly complicated inductive definitions. The problem arises because in a prime event structure each event has the property that it causally depends on a *unique* set of events—it is enabled in a unique way. There are situations where this property does not arise naturally. Consider the event of typing a second character on a line, say "b" to be precise. It depends on having typed a first character, and any character "a", "b", "c" *etc.* will do. The occurrence of typing "b" in the second space cannot be said to causally depend on any particular one occurring in the first place. The only way to describe such a situation by a prime event structure is to work with an idea of event different than that which first suggests itself. In this case, instead of a single event, standing for putting "b" in the second place, we could use events called ab, bb, cb *etc.* meaning "b" after "a", "b" after "b", "b" after "c" *etc.* . And as we mentioned, analogous problems arise in defining some constructions like

parallel composition on prime event structures and give some technical difficulties to do with encoding the history of their dependency into the naming of events.

Such difficulties can be avoided by working with more general event structures, which allow an event to be enabled in several different ways. For simplicity we shall not be quite as general as we might be and keep to a binary conflict relation.

3.1 Definition. An *event structure* is a triple $(E, \#, \vdash)$ where:

(i) E is a set of *events.*

(ii) $\#$ is a binary symmetric, irreflexive relation on E, the conflict relation. We shall write Con for the set of finite conflict-free subsets of E, *i.e.* those finite subsets $X \subseteq E$ for which

$$\forall e, e' \in X.\ \neg(e \# e').$$

(iii) $\vdash \subseteq \mathrm{Con} \times E$ is the *enabling* relation which satisfies

$$X \vdash e \ \ \& \ \ X \subseteq Y \in \mathrm{Con} \Rightarrow Y \vdash e.$$

Our intuitive understanding of the conflict relation is the same as before. The enabling relation is expressed in the notion of configuration we adopt for event structures. A configuration is a set of events which have occurred by some stage in a process. According to our understanding of the conflict relation a configuration should be conflict-free. According to our understanding of the enabling relation every event in a configuration should have been enabled by events which have occurred previously. However the chain of enablings should not be infinite but eventually end with events which are enabled by the null set, and so need no events to occur previously.

3.2 Definition. Let $\mathrm{E} = (E, \#, \vdash)$ be an event structure. Define a *configuration* of E to be a subset of events $x \subseteq E$ which is

(i) *conflict-free:* $\forall e, e' \in x.\ \neg(e \# e')$,

(ii) *secured:* $\forall e \in x \exists e_0, \cdots, e_n \in x.\ e_n = e \ \ \& \ \ \forall i \leq n.\ \{e_0, \cdots, e_{i-1}\} \vdash e_i$.

The set of all configurations of an event structure is written as $\mathcal{F}(\mathrm{E})$.

It is helpful to unwrap condition (ii) a little. It says an event e is secured in a set x iff there is a sequence of events $e_0, \cdots, e_n = e$ in x such that

$$\emptyset \vdash e_0,\ \{e_0\} \vdash e_1, \cdots, \{e_0, \cdots, e_{i-1}\} \vdash e_i, \cdots, \{e_0, \cdots, e_{n-1}\} \vdash e_n.$$

We call such a sequence $e_0, e_1, \ldots, e_n = e$ a *securing* for e in x. The following proposition expresses when an event can be added to a configuration to obtain another configuration. We use $X \subseteq_{fin} Y$ to mean X is a finite subset of Y.

3.3 Proposition. *Let $E = (E, \#, \vdash)$ be an event structure. Suppose $x \in \mathcal{F}(E)$ and $e \in E$. Then $x \cup \{e\} \in \mathcal{F}(E)$ iff*

(i) $\forall X \subseteq_{fin} x.\ X \cup \{e\} \in Con$ *and*

(ii) $\exists X \subseteq_{fin} x.\ X \vdash e$.

Each event structure determines a family of subsets of events, the configurations of the event structure. Such families have a simple characterisation.

3.4 Definition. Let F be a family of subsets. Say a subset X of F is *pairwise compatible* iff for all $x, x' \in X$ there is some $z \in$ F with $x, x' \subseteq z$.

3.5 Theorem. *Let E be an event structure. Its configurations $F = \mathcal{F}(E)$ form a set of subsets of E which satisfy*

(i) coherence: If X is a pairwise compatible subset of F then $\bigcup X \in F$,

(ii) finiteness:

$$\forall x \in F \forall e \in x \exists z \in F.\ (z \text{ is finite } \ \&\ \ e \in z \ \ \&\ \ z \subseteq x),$$

(iii) coincidence-freeness:

$$\forall x \in F \forall e, e' \in x.\ e \neq e' \ \Rightarrow\ (\exists y \in F.\ y \subseteq x \ \ \&\ \ (e \in y \Leftrightarrow e' \notin y)).$$ ∎

3.6 Lemma. *Let F be a family of subsets satisfying (i), (ii) and (iii) above. For all $x, y \in F$*

$$x \subset y \Rightarrow \exists e \in y \setminus x.\ x \cup \{e\} \in F.$$

We can use this fact to show any family satisfying (i), (ii) and (iii) above can be got as the family of configurations of an event structure.

3.7 Theorem. *Let F be a family of configurations of a set E. Define a structure $\mathcal{E}(F) = (E, \#, \vdash)$ on E by taking*

$e \# e'$ iff $\forall x \in F.\ e \in x \Leftrightarrow e' \notin x$,

$X \vdash e$ iff X is conflict-free and $\exists x \in F.\ e \in x \ \ \&\ \ x \subseteq X \cup \{e\}$.

If F is a family of configurations then $\mathcal{E}(F)$ is an event structure such that $\mathcal{F}\mathcal{E}(F) = F$.

Notice we do not have $\mathcal{E}\mathcal{F}$(E) and E equal in general for event structures E, and two different event structures can determine the same family

of configurations. However, for two families of configurations F_0 and F_1, if $\mathcal{E}(\mathrm{F}_0) = \mathcal{E}(\mathrm{F}_1)$ then $\mathrm{F}_0 = \mathrm{F}_1$.

A characterisation of the domains which can be obtained as configurations of an event structure $(E, \#, \vdash)$ will be given, though without proofs, which can be found in [W] and [C]. Our concern will be largely with the more restricted class of stable event structures, the domains of which have a much simpler characterisation along the lines of that for elementary event structures.

Certainly, the partial order $(\mathrm{F}(E), \subseteq)$, of an event structure E, will be finitary domain. In addition it satisfies further axioms on domains which involve the covering relation, prime intervals and the equivalence $\sim$ on them. They are:

Axiom C: $x \prec y \ \ \& \ \ x \prec z \ \ \& \ \ y \uparrow z \ \ \& \ \ y \neq z \Rightarrow y \prec y \sqcup z \ \ \& \ \ z \prec y \sqcup z$

Axiom R: $[x, y] \sim [x, y'] \Rightarrow y = y'$

Axiom V: $[x, x'] \sim [y, y'] \ \ \& \ \ [x, x''] \sim [y, y''] \ \ \& \ \ x' \uparrow x'' \Rightarrow y' \uparrow y''$

3.8 Theorem.
(i) Let E be an event structure. The partial order $(F(E), \subseteq)$ is a finitary domain which satisfies axioms C, R and V. Furthermore it is coherent.
(ii) Let D be a finitary domain which satisfies axioms C, R and V. Then there is an event structure E such that $D \cong (F(E), \subseteq)$. Furthermore D is coherent.

Proof. See the thesis [W] or the book [C] for the proof. The verification of (i) is routine. That of (ii) requires the construction of an event structure from the domain—its events are $\sim$-classes of prime intervals. ∎

Consider an event structure consisting of three events a, b, c with empty conflict relation but where the enabling relation is the least such that $\emptyset \vdash a$, $\emptyset \vdash b$, $\{a\} \vdash c$ and also $\{b\} \vdash c$. In this case the set $\{a, b, c\}$ is a configuration. Because the event c is enabled by either of a or c its occurrence can not be said to causally depend on either one or the other or both. We might say that c has been caused by a and b in parallel. We can not ascribe a partial order of causal dependency to the events as they stand. This is not to say that such an event structure is not a legitimate description of any process. It might well be, but not one to which our intuitions about a partial order of causal dependency apply directly. In a great many examples of processes there are no "parallel causes" and a form of causal dependency, local to configurations, can be seen on the events. For them the event structures are *stable* in the following sense:

3.9 Definition. Let $\mathrm{E} = (E, \#, \vdash)$ be an event structure. Say E is *stable* if it satisfies the following axiom

$$X \vdash e \ \& \ Y \vdash e \ \& \ X \cup Y \cup \{e\} \in \mathrm{Con} \Rightarrow X \cap Y \vdash e.$$

The stability axiom ensures that an event in a configuration is enabled in an essentially unique way. Assume e belongs to a configuration x of a stable event structure. Suppose $X \vdash e$ and $X \subseteq x$. Then $X \cup \{e\} \in \mathrm{Con}$—the enabling $X \vdash e$ is consistent. Take

$$X_0 = \bigcap \{Y \mid Y \subseteq X \ \& \ Y \vdash e\}.$$

Because X is finite this is an intersection of a finite number of sets and we see by the stability axiom that $X_0 \vdash e$. Moreover X_0 is the unique minimal subset of X which enables e. More formally, for any event structure, stable or otherwise, we can define the *minimal enabling* relation $\vdash_{min}$ by

$$X \vdash_{min} e \Leftrightarrow X \vdash e \ \& \ (\forall Y \subseteq X.\ Y \vdash e \Rightarrow Y = X).$$

Then for any event structure

$$Y \vdash e \Rightarrow \exists X \subseteq Y.\ X \vdash_{min} e.$$

But for stable event structures we have uniqueness too, at least for consistent enablings:

$$Y \vdash e \ \& \ Y \cup \{e\} \in \mathrm{Con} \Rightarrow \exists! X \subseteq Y.\ X \vdash_{min} e.$$

It follows that for stable event structures

$$X \vdash_{min} e \ \& \ Y \vdash_{min} e \ \& \ X \cup Y \cup e \in \mathrm{Con} \Rightarrow X = Y.$$

Consequently the families of configurations of stable event structures satisfy the following intersection property.

3.10 Theorem. *Let E be a stable event structure. Then its family of configurations $\mathcal{F}(E)$ satisfies*

$$\forall X \subseteq \mathcal{F}(E).\ X \neq \emptyset \ \& \ X \uparrow \Rightarrow \bigcap X \in \mathcal{F}(E).$$

3.11 Definition. Say a family of sets F is *stable* when it satisfies the following axiom (in addition to those in theorem 3.5)

(stability) $$\forall X \subseteq \mathrm{F}.\ X \neq \emptyset \ \ \& \ \ X\uparrow \Rightarrow \bigcap X \in \mathrm{F}.$$

Thus the configurations of a stable event structure form a stable family. For a stable family there is a partial order of causal dependency on each configuration of events.

3.12 Definition. Let F be a stable family of configurations. Let x be a configuration. For $e, e' \in x$ define

$$e' \leq_x e \Leftrightarrow \forall y \in \mathrm{F}.\ e' \in y \ \ \& \ \ y \subseteq x \Rightarrow e \in y.$$

When $e \in x$ define

$$\lceil e \rceil_x = \bigcap \{y \in \mathrm{F} \mid e \in y \ \ \& \ \ y \subseteq x\}.$$

We say a set y is $\leq_x$-left closed when it satisfies

$$e' \leq_x e \ \ \& \ \ e \in y \Rightarrow e' \in y.$$

As usual, we write $e' <_x e$ for $e \leq_x e' \ \ \& \ \ e \neq e'$.

3.13 Proposition. *Let x be a configuration of a stable family F. Then $\leq_x$ is a partial order and $\lceil e \rceil_x$ is a configuration such that*

$$\lceil e \rceil_x = \{e' \in x \mid e' \leq_x e\}.$$

Moreover the configurations $y \subseteq x$ are exactly the left–closed subsets of $\leq_x$.

Let x be a configuration of a stable family. Intuitively an event e in x can only occur once all its predecessors $\{e' \in x \mid e' <_x e\}$ have occurred.

A special form of stable event structure is obtained from a prime event structure $(E.\#, \leq)$ in the following way: Define

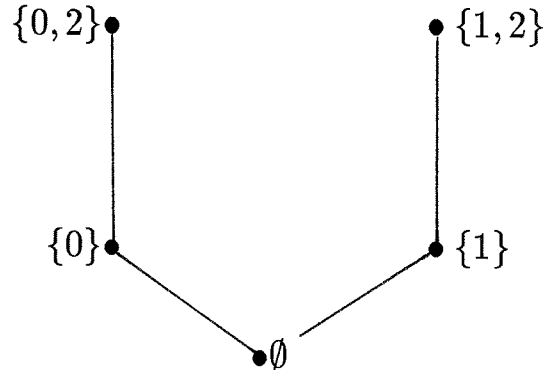
$$X \vdash e \Leftrightarrow \{e\} \subseteq X \cup \{e\}.$$

Then $(E, \#, \vdash)$ is a stable event structure such that $\mathcal{F}(E, \#, \vdash) = \mathcal{L}(E.\#, \leq)$. For such stable families all the orders $\leq_x$, for a configuration x, are restrictioons of a common causal dependency relation $\leq$. This is not the case for stable families in general.

3.14 Example. Let E be the event structure with events $\{0,1,2\}$ with conflict relation the least one such that $0\#1$, and enabling relation the least one such that

$$\emptyset \vdash 0,\ \emptyset \vdash 1,\ \{0\} \vdash 2,\ \{1\} \vdash 2.$$

Then E is a stable event structure and the configurations $\mathcal{F}(\mathrm{E})$ have the form



Let $x = \{0,2\}$ and $y = \{1,2\}$ be particular configurations. Then $0 \leq_x 2$ and $1 \leq_y 2$ but $0 \not\leq_y 2$ and $1 \not\leq_x 2$. The orderings $\leq_x$ and $\leq_y$ are not restrictions of a "global" partial order on events.

3.14 Theorem.

Let E be a stable event structure. Then its family of configurations $\mathcal{F}(E)$ is stable.

Let F be a stable family of configurations. Then $\mathcal{E}(F)$ is a stable event structure.

Families of configurations of stable event structures are prime algebraic. The axiom of stability on event structures has as its counterpart the axiom of distributivity on domains.

3.15 Theorem. *Let F be a family of sets which satisfies (i), (ii), (iii) in theorem 3.5 and is stable (3.11). The partial order $(F,\subseteq)$ is a finitary, coherent, prime algebraic domain; the complete primes are the set $\{\lceil e\rceil_x \mid e \in x \ \&\ x \in \mathcal{F}(E)\}$.*

Referring to theorem 2.12:

3.16 Corollary. *Let E be a stable event structure. The domain of configurations $(\mathcal{F}(E),\subseteq)$ is a coherent, distributive domain.*

Thus stability of event structures appears as distributivity of the domains of configurations. The fact that events must be secured in configurations, expressing the intuition that an event's occurrence can only depend on a

finite number of previous occurrences, reappears as the fact that domains of configurations are finitary.

Conversely, given a finitary, coherent, prime algebraic domain D we can generate an isomorphic stable family *viz.* the family $\mathcal{LPr}(D)$ got by taking the configurations of the prime event structure corresponding to D. Hence we can produce prime, and stable, event structures with domains of configurations isomorphic to D.

4. A complete partial order of event structures.

There is an ordering on event structures which is useful for giving meaning to recursively defined event structures. The order is based on an idea of substructure.

4.1 Definition. Let $\mathrm{E}_0 = (E_0, \#_0, \vdash_0)$ and $\mathrm{E}_1 = (E_1, \#_1, \vdash_1)$ be event structures. Define

$$\begin{aligned}\mathrm{E}_0 \trianglelefteq \mathrm{E}_1 \Leftrightarrow & E_0 \subseteq E_1, \\ & \forall e, e'.\ e \#_0 e' \Leftrightarrow e, e' \in E_0 \ \ \& \ \ e \#_1 e' \quad \text{and} \\ & \forall X, e.\ X \vdash_0 e \Leftrightarrow X \subseteq E_0 \ \ \& \ \ e \in E_0 \ \ \& \ \ X \vdash_1 e.\end{aligned}$$

In this case say E_0 is a *substructure* of E_1.

The notion of substructure is closely tied to that of restriction, an important operation in its own right.

4.2 Definition. Let $\mathrm{E} = (E, \#, \vdash)$ be an event structure. Let $A \subseteq E$. Define the *restriction* of E to A to be

$$\mathrm{E}\lceil A = (A, \#_A, \vdash_A)$$

where

$$\begin{aligned}& X \in \mathrm{Con}_A \Leftrightarrow X \subseteq A \ \ \& \ \ X \in \mathrm{Con}, \\ & X \vdash_A e \Leftrightarrow X \subseteq A \ \ \& \ \ e \in A \ \ \& \ \ X \vdash e.\end{aligned}$$

4.3 Proposition. *Let $E = (E, \#, \vdash)$ be an event structure. Let $A \subseteq E$. Then $E\lceil A$ is an event structure.*

Let $E_0 = (E_0, \#_0, \vdash_0)$ and $E_1 = (E_1, \#_1, \vdash_1)$ be event structures. Then

$$E_0 \trianglelefteq E_1 \Leftrightarrow E_0 = E_1\lceil E_0.$$

If $E_0 \trianglelefteq E_1$ *and* $E_0 = E_1$ *then* $E_0 = E_1$.

Proof. Obvious from the definitions. ∎

This definition of substructure almost gives a complete partial order (cpo) of event structures. There is a least event structure, the unique one with the empty set of events. Each ω-chain of event structures, increasing with respect to $\trianglelefteq$ has a least upper bound, with events, consistency and enabling relations the union of those in the chain. But of course event structures form a class and not a set and for this reason alone they do not quite form a cpo. We call structures like cpos but on a class rather than a set *large cpos*. This is all we need. (Very similar approaches for solving domain equations, or equations for structures like domains, occur elsewhere.)

4.4 Theorem. *The relation* $\trianglelefteq$ *is a partial order on event structures. It has a least event structure* $\underline{\emptyset} =_{def} (\emptyset, \{\emptyset\}, \emptyset)$. *An* ω*–chain of event structures* $E_0 \trianglelefteq E_1 \cdots \trianglelefteq E_n \trianglelefteq \cdots$ *where* $E_n = (E_n, \#_n, \vdash_n)$ *has a least upper bound*

$$\bigcup_{n\in\omega} E_n = (\bigcup_{n\in\omega} E_n, \bigcup_{n\in\omega} Con_n, \bigcup_{n\in\omega} \vdash_n).$$

Proof. Routine. ∎

The substructure relation on event structures is closely related to the rigid embeddings of Kahn and Plotkin [KP].

4.5 Definition. Let D_0 and D_1 be domains. Let $f : \mathrm{D}_0 \to \mathrm{D}_1$ be a continuous function. Say f is an *embedding* iff there is a continuous function $g : \mathrm{D}_1 \to \mathrm{D}_0$, called a *projection*, such that

$$gf(d) = d \text{ for all } d \in D_0 \text{ and}$$
$$fg(c) \sqsubseteq c \text{ for all } c \in D_1.$$

Say f is a *rigid embedding* iff it is an embedding with projection g such that

$$c \sqsubseteq f(d) \Rightarrow fg(c) = c$$

for all $d \in D_0, c \in D_1$.

4.6 Proposition. *Let* E_0 *and* E_1 *be event structures such that* $E_0 \trianglelefteq E_1$. *The inclusion map* $i : \mathcal{F}(E_0) \to \mathcal{F}(E_1)$ *is a rigid embedding with projection* $j : \mathcal{F}(E_1) \to \mathcal{F}(E_0)$ *given by* $j(y) = \bigcup\{x \in \mathcal{F}(E_0) \mid x \subseteq y\}$ *for* $y \in \mathcal{F}(E_1)$.

The next lemma is a great help in proving operations continuous on the large cpo of event structures. Generally it is very easy to show that a unary

operation is monotonic with respect to $\trianglelefteq$ and continuous on the sets of events, a notion we now make precise.

4.7 Definition. Say a unary operation **F** on event structures is *continuous on events* iff for any ω-chain, $\mathrm{E}_0 \trianglelefteq \mathrm{E}_1 \cdots \trianglelefteq \mathrm{E}_n \trianglelefteq \cdots$, each event of $\mathrm{F}(\bigcup_i \mathrm{E}_i)$ is a event of $\bigcup_i \mathrm{F}(\mathrm{E}_i)$.

4.8 Lemma. *Let F be a unary operation on event structures. Then F is continuous iff F is monotonic with respect to $\trianglelefteq$ and continuous on events.*

4.9 Definition. Let **D** be a large cpo ordered by $\trianglelefteq$, with least upper bounds $\bigcup X$ when they exist. Let F be a continuous operation on **D**. Define *fix* F to be the least upper bound

$$\bigcup_{n\in\omega} \mathrm{F}^n(\underline{\emptyset}).$$

4.10 Proposition. *For the situation in the above definition, the element fix F of* **D** *is the least fixed point of F.*

As a simple example of a recursively defined event structure we consider the fixed point of an operation called *prefixing* (sometimes called lifting, or guarding) whose effect on an event structure is to adjoin an extra initial event. Then once it has occurred the behaviour resumes as that of the original event structure.

4.11 Definition. Let a be an event. For an event structure $\mathrm{E} = (E, \#, \vdash)$ define $a\mathrm{E}$ to be the event structure $(E', \#', \vdash')$ where

$$E' = \{(0,a)\} \cup \{(1,e) \mid e \in E\},$$
$$e'_0 \#' e'_1 \Leftrightarrow \exists e_0, e_1.\ e'_0 = (1,e_0)\ \&\ e'_1 = (1,e_1)\ \&\ e_0 \# e_1,$$
$$X \vdash' e' \Leftrightarrow e' = (0,a)\ \text{ or }\ [e' = (1,e_1)\ \&\ (0,a) \in X\ \&\ \{e \mid (1,e) \in X\} \vdash e_1].$$

4.12 Proposition. *For any event a the operation $a(\ )$ is $\trianglelefteq$–continuous on event structures. The least fixed point fix $a(\ )$ has events in 1-1 correspondence with strings in the regular language 1^*0a; any finite subset of events is consistent and the enabling relation satisfies*

$$\emptyset \vdash 0a,$$
$$X \vdash 1^n 0a \Leftrightarrow \{0a, \cdots, 1^{n-1}0a\} \subseteq X,$$

for $n \geq 1$.

Only fixed points of unary operators on event structures have been considered so far. The generalisation to n-ary operators is straightforward following the scheme familiar from domain theory. Such operators are continuous iff they are continuous in each argument separately. They compose to give other continuous operators and we can take their least fixed points to deal with simultaneous recursive definitions. Note that other orderings can sometimes work just as well to handle recursive definitions; for example, simple coordinatewise inclusion gives a large cpo with respect to which most operations are continuous, though it will not work for function space constructions like those mentioned in section 7.

5. Semantics of communicating processes.

One use of event structures is to give a denotational semantics of a language of parallel processes which reflects the parallelism in processes as causal independence between events. The nature of the events, how they interact with the environment, is specified in the language by associating each event with a label from the synchronisation algebra L. The language we shall use is one where processes communicate by events of synchronisation with no value passing. Its syntax has the form:

$$p ::= nil \mid \alpha p \mid p_0 + p_1 \mid p_0 \times p_1 \mid p\lceil\Lambda \mid p[\Xi] \mid x \mid recx.p$$

where x is in some set of variables X over processes, α is a label, Λ is a subset of labels, in $p[\Xi]$ the symbol Ξ denotes a relabelling function between tow sets of labels.

Informally, the product $p_0 \times p_1$ is a form of parallel composition which introduces arbitrary events of synchronisation between processes. Unwanted synchronisations can be restricted away with the help of the restriction operation $p\lceil\Lambda$ and then existing events renamed with the relabelling operation $p[\Xi]$. So in this way we can define specialised parallel compositions of the kind that appear in CCS and CSP, for example.

To explain formally the behaviour of the constructs in the language we describe them as constructions on labelled event structures, so a closed process term in this language is to denote a stable event structure but where the events are labelled.

5.1 Definition. A *labelled event structure* consists of $(E, \#, \vdash, L, l)$ where $(E, \#, \vdash)$ is an event structure, L is a set of labels, not including the element $*$, and l is a function $l : E \to L$ from its events to its labels.

Remark. The special role of the element $*$ will become clear soon.

It shortens some definitions if we use the reflexive conflict relation:

5.2 Notation. In an event structure we shall write $\mathbb{W}$ for the reflexive conflict relation by which we mean that $e\mathbb{W}e'$ in an event structure iff either $e\#e'$ or $e = e'$. With this notation instead of describing the conflict-free sets of an event structure as those sets X such that

$$\forall e, e' \in X.\ \neg(e\#e')$$

we can say they are those sets X for which

$$\forall e, e' \in X.\ e\mathbb{W}e' \Rightarrow e = e'.$$

The term *nil* represents the *nil* process which has stopped and refuses to perform any event; it will denoted by the empty labelled event structure $(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$—no events, no labels.

A *prefixed* process αp first performs an event of kind α to become the process p. Its denotation is given using the prefixing construction of the last section.

5.3 Definition. Let (E, L, l) be a labelled event structure. Let α be a label. Define $\alpha(\mathrm{E}, L, l)$ to be the labelled event structure $(\alpha E, L', l')$ with labels

$$L' = \{\alpha\} \cup L$$

and

$$l'(e') = \begin{cases} \alpha & \text{if } e = (0, \alpha) \\ l(e) & \text{if } e = (1, e) \end{cases}$$

for all $e' \in E'$.

The configurations of $\alpha\mathrm{E}$, a prefixed labelled event structure, have the simple and expected characterisation. (By $\mathcal{F}(\mathrm{E})$ of a labelled event structure E we shall understand the set of configurations of the underlying event structure.)

5.4 Proposition. *Let E be a labelled event structure. Let α be a label.*

$$x \in \mathcal{F}(\alpha E) \Leftrightarrow x = \emptyset \ \ or \ \ [(0,0) \in x \ \ \&\ \ \{e \mid (1,e) \in x\} \in \mathcal{F}(E)].$$

A *sum* $p_0 + p_1$ behaves like p_0 or p_1; which branch of a sum is followed will often be determined by the context and what kinds of events the process is restricted to.

5.5 Definition. Let $\mathrm{E}_0 = (E_0, \#_0, \vdash_0, L_0, l_0)$ and $\mathrm{E}_1 = (E_1, \#_1, \vdash_1, L_1, l_1)$ be labelled event structures. Their *sum*, $\mathrm{E}_0 + \mathrm{E}_1$, is defined to be the structure $(E, \#, \vdash, l)$ with
events $E = \{(0,e) \mid e \in E_0\} \cup \{(0,e) \mid e \in E_1\}$, the disjoint union of sets E_0 and E_1, with injections $\iota_k : E_k \to E$, given by $\iota_k(e) = (k,e)$, for $k = 0, 1$,
conflict relation

$$\begin{aligned} e\#e' \Leftrightarrow & \exists e_0, e_0'.\ e = \iota_0(e_0)\ \&\ e' = \iota_0(e_0')\ \&\ e_0 \#_0 e_0' \\ & \text{or}\ \exists e_1, e_1'.\ e = \iota_1(e_1)\ \&\ e' = \iota_1(e_1')\ \&\ e_1 \#_1 e_1' \\ & \text{or}\ \exists e_0, e_1.\ (e = \iota_0(e_0)\ \&\ e' = \iota_1(e_1))\ \text{or}\ (e' = \iota_0(e_0)\ \&\ e = \iota_1(e_1)) \end{aligned}$$

and enabling relation

$$\begin{aligned} X \vdash e \Leftrightarrow & X \in \mathrm{Con}\ \&\ e \in E\ \& \\ & [(\exists X_0 \in \mathrm{Con}_0, e_0 \in E_0.\ X = \iota_0 X_0\ \&\ e = \iota_0(e_0)\ \&\ X_0 \vdash_0 e_0)\ \text{or} \\ & (\exists X_1 \in \mathrm{Con}_1, e_1 \in E_1.\ X = \iota_1 X_1\ \&\ e = \iota_1(e_1)\ \&\ X_1 \vdash_1 e_1)]. \end{aligned}$$

Its set of labels is the disjoint union $L_0 \uplus L_1$, with injections $\kappa_i : L_i \to L_0 \uplus L_1$ for $i = 0, 1$. Its labelling function acts so

$$l(e) = \begin{cases} \kappa_0(l_0(e_0) & \text{if } e = \iota_0(e_0)) \\ \kappa_1(l_1(e_1) & \text{if } e = \iota_1(e_1)). \end{cases}$$

The choice to take disjoint sets of labels is somewhat arbitrary, though later it does lead to a direct categorical characterisation, and afterall we can obtain another form of sum where copies of events keep their original labels by using relabelling.

The configurations of a sum are obtained from copies of the configurations of the components identified at their empty configurations.

5.6 Proposition. *Let E_0 and E_1 be labelled event structures.*

$$x \in \mathcal{F}(E_0 + E_1) \Leftrightarrow (\exists x_0 \in \mathcal{F}(E_0).\ x = \iota_0 x_0)\ \text{ or }\ (\exists x_1 \in \mathcal{F}(E_1).\ x = \iota_1 x_1).$$

For purposes like modelling value-passing it can be useful to generalise the definition of sum to indexed families of event structures. The definition is straightforward (it can be found in [W1]).

A *product* process $p_0 \times p_1$ behaves like p_0 and p_1 set in parallel. Their events of synchronisation are those pairs of events (e_0, e_1), one from each process; if e_0 is labelled α_0 and e_1 is labelled α_1 the synchronisation event is then labelled (α_0, α_1). Events need not synchronise however; an event in one component may not synchronise with any event in the other. We shall use events of the form $(e_0, *)$ to stand for the occurrence of an event e_0 from one component unsynchronised with any event of the other. Such an event will be labelled by $(\alpha_0, *)$ where α_0 is the original label of e_0 and $*$ is a sort of undefined.

In fact we shall often want to take the first or second coordinates of such pairs and, of course, this could give the value $*$ which we think of as undefined, so that, in effect, we are working with partial functions with $*$ understood to be undefined. We can keep expressions tidier by adopting some conventions about how to treat this undefined value when it appears in expressions and assertions.

5.7 Notation. We shall be working with partial functions θ on events. We indicate that θ is a partial function from E_0 to E_1 by writing $\theta : E_0 \to_* E_1$. Then it may not be the case that $\theta(e)$ is defined and we use $*$ to represent undefined, so $\theta(e) = *$ means the same as $\theta(e)$ is undefined. It is a nuissance when using predicates like $\theta(e) \in X$ to always have to say "provided $\theta(e)$ is defined". Instead we adopt the convention that the basic predicates of equality, membership, conflict, and reflexive conflict are strict in the sense that if they mention $\theta(e)$ this implies $\theta(e)$ is defined. Under this convention, for example,

$$\theta(e) \in X \Rightarrow \theta(e) \text{ is defined, and}$$
$$\theta(e) = \theta(e') \Rightarrow \theta(e) \text{ is defined } \ \& \ \theta(e') \text{ is defined.}$$

We adopt a similar strict interpretation for function application. So if f is a function applied to some value, denoted by a, then $f(a)$ is undefined (gives $*$) if a is undefined.

As usual we represent the image of a set under a partial function by

$$\theta X = \{\theta(e) \mid e \in X \ \ \& \ \ \theta(e) \text{ is defined}\}.$$

5.8 Definition. Let $\mathrm{E}_0 = (E_0, \#_0, \vdash_0, L_0, l_0)$ and $\mathrm{E}_1 = (E_1, \#_1, \vdash_1, L_1, l_1)$ be labelled event structures. Define their *product* $\mathrm{E}_0 \times \mathrm{E}_1$ to be the structure $\mathrm{E} = (E, \#, \vdash, L, l)$ consisting of

events E of the form

$$E_0 \times_* E_1 = \{(e_0, *) \mid e_0 \in E_0\} \cup \{(*, e_1) \mid e_1 \in E_1\} \cup \{(e_0, e_1) \mid e_0 \in E_0 \ \& \ e_1 \in E_1\},$$

with projections $\pi_i : E \to_* E_i$, given by $\pi_i(e_0, e_1) = e_i$, for $i = 0, 1$,
reflexive conflict relation $\mathbb{W}$ given by

$$e \mathbb{W} e' \Leftrightarrow \pi_0(e) \mathbb{W}_0 \pi_0(e') \ \text{ or } \ \pi_1(e) \mathbb{W}_1 \pi_1(e')$$

for all e, e'—we use Con for the conflict-free finite sets,
enabling relation $\vdash$ given by

$$X \vdash e \Leftrightarrow X \in \mathrm{Con} \ \& \ e \in E \ \&$$
$$(\pi_0(e) \text{ is defined} \Rightarrow \pi_0 X \vdash_0 \pi_0(e)) \ \& \ (\pi_1(e) \text{ is defined} \Rightarrow \pi_1 X \vdash_1 \pi_1(e))$$

Its set of labels is

$$L_0 \times_* L_1 = \{(\alpha_0, *) \mid \alpha_0 \in L_0\} \cup \{(*, \alpha_1) \mid \alpha_1 \in L_1\} \cup \{(\alpha_0, \alpha_1) \mid \alpha_0 \in L_0 \ \& \ \alpha_1 \in L_1\}$$

with projections $\lambda_i : E \to_* E_i$, given by $\lambda_i(\alpha_0, \alpha_1) = \alpha_i$, for $i = 0, 1$. Its labelling function is defined to act on an event e so

$$l(e) = (l_0 \pi_0(e), l_1 \pi_1(e)).$$

We characterise the configurations of the product of two event structures in terms of their configurations.

5.9 Proposition. *Let $E_0 \times E_1$ be the product of labelled event structures with projections π_0, π_1. Let $x \subseteq E_0 \times_* E_1$, the events of the product. Then $x \in \mathcal{F}(E_0 \times E_1)$ iff*

$\pi_0 x \in \mathcal{F}(E_0)$ & $\pi_1 x \in \mathcal{F}(E_1)$,
$\forall e, e' \in x.\ \pi_0(e) = \pi_0(e')$ or $\pi_1(e) = \pi_1(e') \Rightarrow e = e'$,
$\forall d \in x \exists y \subseteq x.\ \pi_0 y \in \mathcal{F}(E_0)$ & $\pi_1 y \in \mathcal{F}(E_1)$ & $e \in y$ & $|y| < \infty$ and
$\forall e, e' \in x.\ e \neq e' \Rightarrow \exists y \subseteq x.\ \pi_0 y \in \mathcal{F}(E_0)$ & $\pi_1 y \in \mathcal{F}(E_1)$ & $(e \in y \Leftrightarrow e' \notin y)$.

The proposition above expresses the intuition that an allowable behaviour of the product of two processes is precisely that which "projects" to allowable

behaviours in the component processes—the complicated-looking conditions (c) and (d) are there just to ensure that the family of sets is finitary and coincidence-free.

The *restriction* $t\lceil\Lambda$ behaves like the process p but with its events restricted to those with labels which lie in the set Λ.

5.10 Definition. Let $\mathrm{E} = (E, \#, \vdash, L, l)$ be a labelled event structure. Let Λ be a subset of labels. Define the restriction $\mathrm{E}\lceil\Lambda$ to be $(E', \#', \vdash', L \cap \Lambda, l')$ where $(E', \#', \vdash')$ is the restriction of $(E, \#, \vdash)$ to the events $\{e \in E \mid l(e) \in \Lambda\}$ and the labelling function l' is the restriction of the original labelling function to the domain $L \cap \Lambda$.

5.11 Proposition. *Let $E = (E, \#, \vdash, L, l)$ be a labelled event structure. Let $\Lambda \subseteq L$.*

$$x \in \mathcal{F}(E\lceil\Lambda) \Leftrightarrow x \in \mathcal{F}(E) \ \ \& \ \ \forall e \in x.\ l(e) \in \Lambda.$$

A relabelled process $p[\Xi]$ behaves like p but with the events relabelled according to Ξ.

5.12 Definition. Let $\mathrm{E} = (E, \#, \vdash, L, l)$ be a labelled event structure. Let Λ, L' be sets of labels and $\Xi : \Lambda \to L'$. Define the relabelling $\mathrm{E}[\Xi]$ to be $(E, \#, \vdash, L', l')$ where

$$l'(e) = \begin{cases} \Xi l(e) & \text{if } l(e) \in \Lambda, \\ l(e) & \text{otherwise.} \end{cases}$$

In order to give a meaning to the recursively defined processes of the form $recx.p$ we use the fact that the operations are continuous with respect to a large c.p.o. of labelled event structures. The large c.p.o. of event structures $\trianglelefteq$ extends naturally to labelled event structures in such a way that operations like parallel composition are continuous.

Define the ordering $\trianglelefteq_L$ on labelled event structures by:

$$(\mathrm{E}_0, L_0, l_0) \trianglelefteq_L (\mathrm{E}_1, L_1, l_1) \Leftrightarrow \mathrm{E}_0 \trianglelefteq \mathrm{E}_1 \ \ \& \ \ L_0 \subseteq L_1 \ \ \& \ \ l_0 = l_1\lceil E_0.$$

The null labelled event structure $(\underline{\emptyset}, \emptyset, \emptyset)$ is the least L-labelled event structure with respect to $\trianglelefteq_L$. Of course, $\trianglelefteq_L$ has least upper bounds of ω-chains; the lub of a chain $(\mathrm{E}_0, L_0, l_0), \ldots, (\mathrm{E}_n, L_n, l_n), \ldots$ takes the form $(\bigcup_n \mathrm{E}_n, \bigcup_n L_n, \bigcup_n l_n)$. All the operations prefixing, sum, restriction, relabelling and parallel composition are continuous with respect to $\trianglelefteq_L$. The proofs follow by applying lemma

4.8. It is straightforward to check that each operation is monotonic and continuous on events for each argument separately, and so is $\trianglelefteq$–continuous. Thus we can give a denotational semantics to $\mathbf{Proc}_L$ by representing recursively defined processes as the least fixed points of continuous operation.

5.13 Definition. *Denotational semantics:* Define an *environment* for process variables to be a function ρ from process variables X to labelled event structures. For a term t and an environment ρ, define the denotation of t with respect to ρ written $[\![t]\!]\rho$ by the following structural induction—syntactic operators appear on the left and their semantic counterparts on the right.

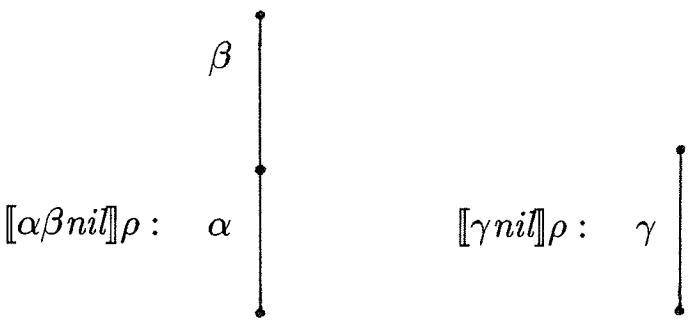
$$\begin{array}{ll} [\![nil]\!]\rho & =(\underline{\emptyset},\emptyset) \\ [\![x]\!]\rho & =\rho(x) \\ [\![\alpha t]\!]\rho & =\alpha([\![t]\!]\rho) \\ [\![t_1+t_2]\!]\rho & =[\![t_1]\!]\rho+[\![t_2]\!]\rho \end{array} \qquad \begin{array}{ll} [\![t\lceil\Lambda]\!]\rho & =[\![t]\!]\rho\lceil\Lambda \\ [\![t[\Xi]]\!]\rho & =[\![t]\!]\rho[\Xi] \\ [\![t_1\times t_2]\!]\rho & =[\![t_1]\!]\rho\times[\![t_2]\!]\rho \\ [\![recx.t]\!]\rho & =fix\ \Gamma \end{array}$$

where Γ is an operation on labelled event structures given by $\Gamma(\mathrm{E})=[\![t]\!]\rho[\mathrm{E}/x]$ and *fix* is the least–fixed–point operator.

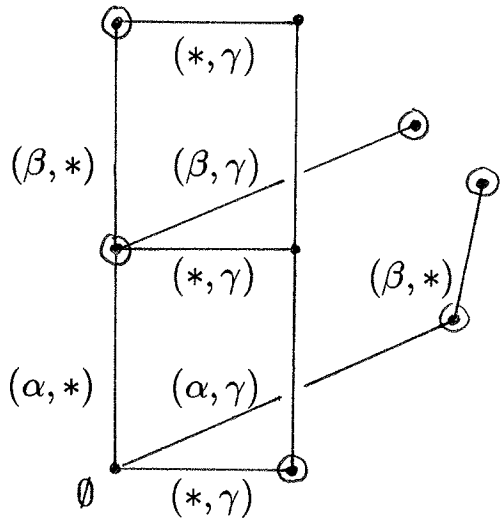
Remark. A straightforward structural induction shows that Γ above is indeed continuous with respect to $\trianglelefteq_L$ so the denotation of a recursively defined process is really the least fixed point of the associated functional Γ.

It can be shown that each operation preserves the stability axiom on event structures and so restricts to an operation on stable event structures. Consequently, according to the denotational semantics each closed process term denotes a labelled stable event structure. By taking events to be complete primes in the families of configurations—using the operation $\mathcal{P}r$—we obtain labelled prime event structures as denotations.

5.14 Example. With respect to any environment ρ, the terms $\alpha\beta nil$ and γnil denote event structures with domains of configurations of the form below:



where we label prime intervals by the labels of the corresponding events. The domains of configurations of $[\![\alpha\beta nil \times \gamma nil]\!]\rho$ look like:



where we have encircled the complete primes. A labelled prime event structure is obtained by taking the complete primes as new events and the causal dependency and conflict relations as restrictions of the ordering and incompatibility relations. (We leave this to the reader.)

Labelled stable event structures are fairly complicated things, and constructions on them are complicated too. Originally, for the most part, languages like the one we are using were given an "interleaving semantics" in the sense that parallelism was simulated by nondeterministically interleaving the events of two processes set in parallel, except where they were synchronised together. How are we to check that our semantics agrees with such semantics even though it expresses more about the computations?

Part of the answer lies in taking a more abstract view of the constructions used by employing elementary category theory. Category theory not only provides abstract characterisations of constructions like product, restriction and relabelling, which determine the constructions to within isomorphism, but also provides techniques for proving the consistency of one semantics with respect to another.

Examples of morphisms have been introduced implicitly in the definition of sum and product, and, for example, the definition of product used projection functions on the events and labels. Through them, proposition 5.9 formalises the intuition that an allowable behaviour of the product of two processes is precisely that which "projects" to allowable behaviours in the component processes.

The following gives the general definition of morphism between labelled event structures.

5.15 Definition. Let $\mathrm{E}_0 = (E_0, \#_0, \vdash_0, L_0, l_0)$ and $\mathrm{E}_1 = (E_1, \#_1, \vdash_1, L_1, l_1)$ be labelled event structures. A *(partially synchronous) morphism* from E_0 to E_1 is a pair (η, λ) of partial functions $\eta : E_0 \rightarrow_* E_1$ on events and $\lambda : L_0 \rightarrow_* L_1$ on labels which satisfies

(i) $\eta(e) \mathbb{W}_1 \eta(e') \Rightarrow e \mathbb{W}_0 e'$,
(ii) $X \vdash_0 e \ \ \& \ \ \eta(e)$ is defined $\Rightarrow \eta X \vdash_1 \eta(e)$ and
(iii) $l_1 \eta = \lambda l_0$.

(Note by our strictness convention in handling undefined the truth of $\eta(e) \mathbb{W}_1 \eta(e')$ asserts also that $\eta(e)$ and $\eta(e')$ are defined.)

Remark. The partially synchronous morphisms here are are a little different from those presented in [W1]. Firstly, because conflict is based on a binary relation in these notes, the conditions are expressed differently—though with the same effect on configurations. Secondly, here event structures carry labels and we have made the morphisms respect them. While this addition does permit a categorical characterisation of restriction and relabelling, it does not complicate many of the proofs.

A morphism $(\eta, \lambda) : \mathrm{E}_0 \rightarrow \mathrm{E}_1$ between labelled event structures E_0 and E_1 expresses how behaviour in E_0 determines behaviour in E_1. The partial function η expresses how the occurrence of events in E_0 imply the simultaneous occurrence of events in E_1; the fact that $\eta(e_0) = e_1$ means that the event e_1 is a component of the event e_0 and, in this sense, that the occurrence of e_0 implies the simultaneous occurrence of e_1. The final condition (iii) simply ensures the agreement of the functions on events and the function on labels. To understand condition (i) it is helpful to break it into an equivalent conjunction of the following two conditions:

(ia) $\eta(e) = \eta(e') \Rightarrow e \mathbb{W}_0 e'$,
(ib) $\eta(e) \#_1 \eta(e') \Rightarrow e \#_0 e'$

for all events e, e' in E_0. The condition (ia) says if two distinct events e, e' have the same image then they are in conflict. This formalises the idea that if two distinct events in E_0 have the same image e_1 in E_1 then they cannot belong to the same configuration. Otherwise, by the property of coicidence-freeness of configurations, there would be a configuration containing both events but with a subconfiguration which separated them. Considering the effect of the occurrence of the two events, under the morphism, this would lead to the contradiction of e_1 occurring twice. Condition (1b) says that if two events

have images in conflict then they must themselves be in conflict. It says conflict-free sets in E_0 determine conflict-free behaviour in E_1. Condition (ii) in the definition says that a morphism preserves enabling. Together conditions (i) and (ii) ensure that a configuration in E_0 determines a configuration in E_1, in other words that morphisms preserve configurations.

5.16 Proposition. *Let $(\eta, \lambda) : E_0 \to E_1$ be a morphism of stable event structures. Then*

$$x \in \mathcal{F}(E_0) \Rightarrow (\eta x \in \mathcal{F}(E_1) \ \ \& \ \ \forall e, e' \in x.\ \eta(e) = \eta(e') \Rightarrow e = e').$$

5.17 Proposition. *Labelled event structures with partially synchronous morphisms form a category with composition the coordinatewise composition of partial functions and identity morphisms pairs of identity functions on events and labels.*

The product and sum are familiar categorical constructions:

5.18 Proposition. *The product of labelled event structures is the categorical product in the category of labelled event structures with partially synchronous morphisms. The sum is the categorical coproduct.*

These facts determine the product and sum to within isomorphism.

Restriction and relabelling are constructions which depend on labelling sets and functions between them. Seeing them as categorical constructions involves dealing explicitly with functions on labelling sets and borrowing a couple of fundamental ideas from indexed category theory.

We can project a morphism (η, λ) between labelled event structures to a partial function $p(\eta, \lambda) = \lambda$ on labelling sets, determining a functor p from the category of labelled event structures to the category of labelling sets with functions. Suppose E is an event structure with labelling set L. We will lose no generality if we assume the restricting set Λ is a subset of L. Then there is an inclusion morphism $j : \Lambda \to L$ in the category of labelling sets with partial functions. The restriction $\mathrm{E}\lceil\Lambda$ has labelling set Λ and there is a morphism $(i, j) : \mathrm{E}\lceil\Lambda \to \mathrm{E}$ where i is the inclusion map on events. This morphism is characterised abstractly as the *cartesian lifting* of j with respect to E. A morphism $f : \mathrm{E}' \to \mathrm{E}$ between event structures is said to be a *cartesian lifting* of the morphism $p(f)$ between labelling sets with respect to E if for any morphism $g : \mathrm{E}'' \to \mathrm{E}$ on event structures and morphism $\lambda : p(\mathrm{E}'') \to p(\mathrm{E}')$ between labelling sets for which $p(f)\lambda = p(g)$ there is a unique morphism $h : \mathrm{E}'' \to \mathrm{E}'$ such that $p(h) = \lambda$ and $fh = g$.

Relabelling can be characterised in a similar way but using a dual notion. Suppose E is a labelled event structure with labelling set L. To simplify the explanation, as we lose no generality, we assume $\Xi : L \to L'$ is a total function. The labelled event structure $\mathrm{E}[\Xi]$ has labelling set L' and there is a morphism $(1,\Xi) : \mathrm{E} \to \mathrm{E}[\Xi]$, where 1 is the identity function on the set of events. This morphism is a *cocartesian lifting* of Ξ with respect to E. A morphism $f : \mathrm{E} \to \mathrm{E}'$ between event structures is said to be a *cocartesian lifting* of the morphism $p(f)$ between labelling sets with respect to E if for any morphism $g : \mathrm{E} \to \mathrm{E}''$ on event structures and morphism $\lambda : p(\mathrm{E}') \to p(\mathrm{E}'')$ between labelling sets for which $\lambda p(f) = p(g)$ there is a unique morphism $h : \mathrm{E}' \to \mathrm{E}''$ such that $p(h) = \lambda$ and $hf = g$.

It is not hard to see that the notions of cartesian and cocartesian liftings are general notions which make sense whenever we have a functor p from one category to another. It can be checked that the domain of a cartesian lifting and the codomain of a cocartesian lifting are determined to within isomorphism, and moreover, by isomorphisms which project to the identity.

To summarise:

5.19 Proposition. *Operations of restriction are obtained as cartesian liftings of an inclusion between labelling sets. Operations of relabelling are obtained as cocartesian liftings of total functions between labelling sets.*

I do not know a categorical characterisation of prefixing. Nor do I know a categorical way of expressing hiding (the operation of making certain specified events hidden or internal, so that they can occur within a process but cannot synchronise further with any events in the environment).

The abstract and general constructions of the operations used to describe our language of processes are useful when we come to consider different models. The same abstract constructions apply in many different models. We can draw on some general results about functors preserving constructions in order to show how semantics in terms of one model is preserved when it is translated to semantics in terms of another.

For example, a well-known interleaving model is that of synchronisation trees in which nodes represent states and arcs carrying labels are thought of as labelled events. By regarding such trees as (families of configurations of) special kinds of labelled prime event structures we inherit a notion of partially synchronous morphism on them. In the category of synchronisation trees the coproduct is an operation which "glues" trees together at their roots, while

the product is of the kind one would hope for from Milner's expansion theorem. Again cartesian and cocartesian liftings give restriction and relabelling operations. The identification of synchronisation trees with certain kinds of labelled event structures is essentially an inclusion functor from the category of synchronisation trees to the category of labelled event structures. There is another functor, going the other way from labelled event structures to synchronisation trees, which given a labelled event structure serialises its event occurrences to produce a synchronisation tree whose behaviour is an interleaved version of that of the original event structure. These two functors bear a useful relationship with each other: the serialisation functor is right adjoint to the inclusion functor, a fact which characterises it to within isomorphism. The adjunction is, in fact, of a special form: if we serialise a synchronisation tree we obtain a synchronisation tree isomorphic to the original. This makes the adjunction a coreflection.

Armed with this knowledge we can make use of some general results. Right adjoints preserve all limits, and products in particular. This immediately yields that the serialisation of the product of labelled event structures is the product (as a tree) of their serialisations. There are other useful facts like that left adjoints preserve coproducts, and results on the preservation of cartesian and cocartesian liftings. Such facts are clearly useful in showing how semantics is preserved by operations, like serialisation, across different models. They can save us calculations in another way. To give the idea, rather than work out the product in trees we can see they exist from the existence of a coreflection, given that we know we have products of event structures. If T_0 and T_1 are synchronisation trees, we can regard them as event structures, form their product $T_0 \times_E T_1$ as event structures, serialise this to get $S(T_0 \times_E T_1)$ and know this is the product of $S(T_0)$ and $S(T_1)$ because the right adjoint S preserves products. As S is part of a coreflection $S(T_0) \cong T_0$ and $S(T_1) \cong T_1$. So $S(T_0 \times_E T_1)$ is the product of T_0 and T_1 in the category of synchronisation trees.

For a a detailed account of categories of models for parallel computation see [W1] (although the treatment of labels there is different, the presence of labels does not complicate the proofs much) and [W2] (where labels are taken into account but largely for the model of Petri nets).

6. Nets, traces and event structures.

In this section we sketch the relationship between event structures and two other models of parallel computation, Petri nets and trace languages. All three follow a similar philosophy in the way they represent concurrency.

In defining Petri nets and constructions on them we shall use multisets and their notation—see the appendix for a quick summary. Note, especially, that we call a multiset a *set* when its multiplicities are all less than or equal to 1.

6. Definition. A *Petri net* is a structure $(B, M_0, E, pre, post)$ where,
B is a set of *conditions*,
M_0 is a multiset of conditions, called the *initial marking*,
E is a set of *events*,
pre and *post* are multirelations $E \to_m B$, called the *pre* and *post* condition maps respectively,
which satisfy the restriction:
$\forall b \in B.\ M_{0b} \neq 0$ or $(\exists e \in E.\ pre(e)_b \neq 0$ or $post(e)_b \neq 0)$.

6.2 Notation. When the context makes it clear we shall use ${}^\bullet(\ )$ and $(\ )^\bullet$ for the pre and post condition maps.

The restriction in the definition of Petri nets is there for the technical reason that in the full treatment with morphisms on nets it ensures that we can always manage with multisets in which the multiplicities are finite.

6.3 Definition. The behaviour of nets:
Let $N = (B, M_0, E, pre, post)$ be a Petri net.
A *marking* M is a multiset of conditions, *i.e.* $M \in m(B)$.
Let M, M' be markings. Let A be a finite multiset of events. Define

$$M \xrightarrow{A} M' \Leftrightarrow {}^\bullet A \leq M \ \ \& \ \ M' = M - {}^\bullet A + A^\bullet.$$

This gives the *transition relation* between markings. The transition $M \xrightarrow{A} M'$ means that the finite multiset of events A can occur concurrently from the marking M to yield the marking M'.

A *reachable marking* of N is a marking M for which

$$M_0 \xrightarrow{A_0} M_1 \xrightarrow{A_1} \cdots \xrightarrow{A_{n-1}} M_n = M$$

for some markings and finite multisets of events.

6.4 Notation. We write $M \xrightarrow{A} M' : N$ to mean M is a reachable marking of N and $M \xrightarrow{A} M'$ is a transition of N. We write $M \xrightarrow{A}: N$ to mean $M \xrightarrow{A} M' : N$, for some M'.

6.5 Definition. Say a Petri net N is *safe* iff ${}^\bullet e$ and $e^\bullet$ are sets, for all events e, and whenever $M \xrightarrow{A}: N$ then M and A are sets.

As Mazurkiewicz has shown, the behaviour of safe nets can be analysed through the notion of trace, and we refer the reader to his lecture notes in this volume for a fuller account. (Please be prepared for our definitions to appear superficially different from his.)

6.6 Definition. A *trace language* consists of (A, I, T) where A is a set of *events*, I is a binary, symmetric, irreflexive relation on A called the *independence* relation and T is a nonempty subset of sequences A^* which is

prefix closed: $sa \in T \Rightarrow s \in T$ for all $s \in A^*, a \in A$,

I-closed: $sabt \in T \ \& \ aIb \Rightarrow sbat \in T$ for all $s, t \in A^*, a, b \in A$.

Say a trace language is *coherent* iff

$$sa \in T \ \& \ sb \in T \ \& \ aIb \Rightarrow sab \in T$$

for all $s \in A^*, a, b \in A$.

Probably more familiar is the definition of a trace language as a set of equivalence classes along the following lines.

6.7 Definition. Let (A, I, T) be a trace language. Define the relation $=_I$ on T to be the smallest equivalence relation such that

$$t_0 =_I t_1' \text{ if } \exists s, s' \in T, a, b \in A.\ aIb \ \& \ t_0 = sabs' \ \& \ t_1 = sbas'.$$

For $s \in T$ write $\{s\}_I$ for the $=_I$-equivalence class of s. For $=_I$-equivalence classes x, y, define $x \leq y$ iff there are s, s' such that $x = \{s\}_I$ and $y = \{t\}_I$ and s is a prefix of t.

Write T_I for the set of $=_I$ equivalence classes ordered by $\leq_I$; call its members traces.

Of course, all traces are equivalence classes of finite sequences. They represent finite computations. Not having limit points, traces do not form domains. However, we can add limit points by taking a *completion by ideals*.

6.8 Definition. Let $(P, \leq)$ be a partial order. An *ideal* of P is a left-closed, directed subset. In particular the *principal ideals* are those subsets of the

form $\{q \in P \mid q \leq p\}$, for some $p \in P$. Define the *completion by ideals*, P^∞, to be the partial order of ideals ordered by inclusion.

6.9 Proposition. *Let P be a partial order. The ideal completion P^∞ is a domain iff P has least upper bounds of all finite compatible subsets. In the case where P^∞ is a domain its finite elements are precisely the principal ideals.*

Remark. The proposition holds in further generality, for what are known as algebraic complete partial orders. These can be characterised as those partial orders which arise, to within isomorphism, as ideal completions of partial orders with a least element.

6.10 Lemma. *Let (A, I, T) be a trace language. The order T_I^∞ is a finitary, prime algebraic domain.*

Proof. A proof can be found in Bednarczyk's thesis [B] thoughthere they are given just for full trace languages with all sequences. Bednarczyk's proofs (p.54–58) apply in the wider context of all trace languages, to show that T_I has least upper bounds of finite compatible sets and is distributive *i.e.*

$$x \uparrow y \Rightarrow (x \sqcup y) \sqcap z = (x \sqcap z) \sqcup (y \sqcap z).$$

T_I can thus be identified with the finite elements of a distributive domain T_I^∞. Built out of finite sequences ordered by extension, the domain has to be finitary and hence prime algebraic. ∎

6.11 Lemma. *If a trace language (A, I, T) is coherent then the domain T_I^∞ is coherent.*

Proof. Again, Bednarczyk's proof (p.58) applies more generally. Another proof follows from the representation theorem 3.8. It is easily checked that all the axioms C, R, V hold of the finitary domain T_I^∞ (Notice that if two prime intervals are $\sim$-equivalent then their event symbols from the trace language are the same). The domain T_I^∞ is thus coherent by 3.8(ii). ∎

6.12 Definition. Let N be a safe net. Define $\mathit{Trace}(N) = (A, I, T)$ where A is the set of events of N, $e_0 I e_1$ holds between events e_0, e_1 iff $({}^\bullet e_0 \cup e_0{}^\bullet) \cap ({}^\bullet e_1 \cup e_1{}^\bullet) = \emptyset$, and T consists of the set of firing sequences $\langle e_1, \ldots, e_{n-1} \rangle$ from the initial marking (*i.e.* $M_0 \xrightarrow{e_1} \cdots \xrightarrow{e_{n-1}}$ where M_0 is the initial marking).

6.13 Proposition. *Let N be a safe net. Then $\mathit{Trace}(N)$ is a coherent trace language.*

Proof. Well-known, and fairly easy to see (See [Maz]). ∎

6.14 Corollary. *Let N be a safe net. Let Trace$(N) = (A, I, T)$. Then T_I^∞ is a coherent, finitary prime algebraic domain.*

Thus via traces we have defined an operator from safe nets to coherent, finitary prime algebraic domains. As prime event structures are equivalent to such domains we can as well obtain an operator $\mathcal{E}$ taking a safe net to a prime event structure whose events correspond to occurrences of events in the net.

There is an operator in the opposite direction from prime event structures to safe nets. Given an event structure we can form a Petri net with the same behaviour by adding enough conditions to the events to express the causal dependency and conflict of the original event structure: if $e < e'$ in the event structure then we put in a condition as a precondition of e' and a postcondition of e; if $e\#e'$ we make e and e' share a common precondition. More precisely we can build conditions out of events in accord with the causal dependency and conflict relations in the following way:

6.15 Definition. Let $\mathrm{E} = (E, \#, \leq)$ be an event structure. Define $\mathcal{N}(\mathrm{E})$ to be (B, E, F, M) where

$$M = \{(\emptyset, A) \mid A \subseteq E \ \& \ (\forall a, a' \in A.\ a(\# \cup 1)a')\}$$
$$B = M \cup \{(e, A) \mid e \in E \ \& \ A \subseteq E \ \& \ (\forall a, a' \in A.\ a(\Vdash)a') \ \& \ (\forall a \in A.\ e < a)\}$$
$$F = \{(e, (e, A)) \mid (e, A) \in B\} \cup \{((c, A), e) \mid (c, A) \in B \ \& \ e \in A\}.$$

This shows how we can translate between the different models of safe Petri nets and prime event structures, or equivalent prime algebraic domains. Notice $\mathcal{E}\mathcal{N}(\mathrm{E}) \cong \mathrm{E}$. In fact the scheme extends to the situation when we label events and endow nets with morphisms on the same lines as those for event structures. The operations $\mathcal{E}$ and $\mathcal{N}$ between the two models extend to functors which form a coreflection. The presence of morphisms makes $\mathcal{E}$ less arbitrary than it might otherwise seem—there are many ways of building a net with the same behaviour as a prime event structure (see *e.g.* ch.6 of [W2] for another).

The story extends to many more categories of models. The morphisms on the different models also express synchronisation. The appropriate constructions to give denotations to our language of processes have the same abstract characterisation in all the different categories. The fact that they are preserved in moving from one model to another follows from general properties of the adjoints (see [W1] and [B] for details, though just for unlabelled structures).

7. Higher-type event structures.

So far we have focussed on one particular interpretation of events, as actions of synchronisation between processes. Of course other interpretations are possible. We now look at another specific interpretation in which event structures are used to represent higher types. We have seen how stable event structures represent domains. In fact such domains can be made into a cartesian closed category. Not only does this yield a model for various typed λ-calculi but, using the techniques of section 4, we can produce models of so called untyped λ-calculi. We shall not present any proofs in this section—they can be found in [W1].

As morphisms between stable event structures we take *stable* functions between their domains of configurations.

7.1 Definition. Let E_0 and E_1 be stable event structures. A stable function from E_0 to E_1 is a continuous function $f : (\mathcal{F}(\mathrm{E}_0), \subseteq) \to (\mathcal{F}(\mathrm{E}_1), \subseteq)$ on their configurations which is continuous and satisfies

$$\forall X \subseteq \mathcal{F}(\mathrm{E}_0).\ X \neq \emptyset \ \&\ X\uparrow \Rightarrow f(\bigcap X) = \bigcap f X.$$

7.2 Proposition. *Stable event structures with stable functions composed as functions and the usual identities form a category.*

The key idea to the treatment of higher types is the general construction of an event structure to represent the function space of stable functions between two event structures. We must somehow represent the space of stable, continuous functions $f : \mathrm{E}_0 \to \mathrm{E}_1$ between two stable event structures E_0 and E_1 as an event structure itself. This is done by taking the events of a "function space" event structure to be basic parts of functions (x, e) standing for the event of outputting e at input x, a finite configuration of E_0. The function f will correspond to a configuration of events (x, e) in which x is a minimal input configuration at which e is output.

7.3 Definition. Let $\mathrm{E}_0 = (E_0, \#_0, \vdash_0)$ and $\mathrm{E}_1 = (E_1, \#_1, \vdash_1)$ be stable event structures. Their *stable function space*, $[\mathrm{E}_0 \to \mathrm{E}_1]$ is defined to be the event structure $(E, \#, \vdash)$ with events E consisting of pairs (x, e) where x is a finite configuration in $\mathcal{F}(\mathrm{E}_0)$ and $e \in E_1$, a reflexive conflict relation $\mathbb{W}$ given by

$$(x, e)\mathbb{W}(x', e') \text{ iff } x \uparrow x' \ \&\ e\mathbb{W}_1 e'$$

and an enabling relation given by

$$\{(x_0, e_0), \cdots, (x_{n-1}, e_{n-1})\} \vdash (x, e) \text{ iff } \{e_i \mid x_i \subseteq x\} \vdash_1 e.$$

7.4 Proposition. *The stable function space of two stable event structures is a stable event structure. The stable function space construction is $\trianglelefteq$-continuous.*

The configurations of a stable function space $[\mathrm{E}_0 \to \mathrm{E}_1]$ correspond to stable, continuous functions $\mathcal{F}(\mathrm{E}_0) \to \mathcal{F}(\mathrm{E}_1)$.

7.5 Definition. Let E_0 and E_1 be stable event structures.

For $F \in \mathcal{F}([\mathrm{E}_0 \to \mathrm{E}_1])$ define

$$(\phi(F))(x) = \{e \in E_1 \mid \exists x' \subseteq x.\ (x', e) \in F\}$$

for $x \in \mathcal{F}(\mathrm{E}_0)$.

For $f : \mathcal{F}(\mathrm{E}_0) \to \mathcal{F}(\mathrm{E}_1)$ a stable, continuous function define $\mu(f)$ a subset of events of $[\mathrm{E}_0 \to \mathrm{E}_1]$ by

$$(x, e) \in \mu(f) \Leftrightarrow e \in f(x) \ \ \& \ \ (\forall x' \subseteq x.\ e \in f(x') \Rightarrow x' = x).$$

7.6 Theorem. *Let E_0 and E_1 be stable event structures.*
(i) For $F \in \mathcal{F}([E_0 \to E_1])$, the function $\phi(F) : \mathcal{F}(E_0) \to \mathcal{F}(E_1)$ is continuous and stable.
(i) For $f : \mathcal{F}(E_0) \to \mathcal{F}(E_1)$ a stable, continuous function, the subset $\mu(f) \in \mathcal{F}([E_0 \to E_1])$.
(iii) Further, ϕ and μ are mutual inverses giving a 1-1 correspondence between configurations $\mathcal{F}([E_0 \to E_1])$ and stable, continuous functions $\mathcal{F}(E_0) \to \mathcal{F}(E_1)$.

The product in the category is obtained very simply. The event structures are allowed to operate disjointly, completely in parallel, neither one having an effect on the other. It is easily defined for all event structures not just the stable ones.

7.7 Definition. Let $\mathrm{E}_0 = (E_0, \#_0, \vdash_0)$ and $\mathrm{E}_1 = (E_1, \#_1, \vdash_1)$ be stable event structures. Their *disjoint product*, $\mathrm{E}_0 \oplus \mathrm{E}_1$, is the structure $(E, \#, \vdash)$ where the events are

$$E = \{0\} \times E_0 \cup \{1\} \times E_1,$$

a disjoint union, the consistency predicate is given by

$$X \in \mathrm{Con} \Leftrightarrow \{e \mid (0,e) \in X\} \in \mathrm{Con}_0 \ \ \& \ \ \{e \mid (1,e) \in X\} \in \mathrm{Con}_1,$$

and the enabling by

$$\begin{aligned} X \vdash e \Leftrightarrow & X \in \mathrm{Con} \ \ \& \ \ e \in E \ \ \& \\ & [(\exists e_0 \in E_0.\ e = (0,e_0) \ \ \& \ \ \{e' \mid (0,e') \in X\} \vdash_0 e_0) \ \text{ or} \\ & (\exists e_1 \in E_1.\ e = (1,e_1) \ \ \& \ \ \{e' \mid (1,e') \in X\} \vdash_1 e_1)]. \end{aligned}$$

Define the *projections* $p_k : \mathcal{F}(\mathrm{E}_0 \oplus \mathrm{E}_1) \to \mathcal{F}(\mathrm{E}_k)$ by taking $p_k(x) = \{e \mid (k,e) \in x\}$, for $k = 0, 1$.

7.8 Proposition. *Let E_0 and E_1 be event structures with events E_0, E_1 respectively. Then*

$$x \in \mathcal{F}(E_0 \oplus E_1) \Leftrightarrow x \subseteq E_0 \uplus E_1 \ \ \& \ \ p_0(x) \in \mathcal{F}(E_0) \ \ \& \ \ p_1(x) \in \mathcal{F}(E_1).$$

There is a 1-1 correspondence between $\mathcal{F}(E_0 \oplus E_1)$ and $\mathcal{F}(E_0) \times \mathcal{F}(E_1)$ given by

$$x \mapsto (p_0(x), p_1(x)).$$

The disjoint product is $\trianglelefteq$-continuous.

Thus we can identify x, a configuration of a disjoint product, with the pair $(p_0(x), p_1(x))$.

7.9 Theorem. *The disjoint product $E_0 \oplus E_1$ of stable event structures E_0 and E_1, with projections π_0, π_1, is a product in the category $\mathbf{E}_{stab}$.*

At this point we can directly prove the cartesian closure of $\mathbf{E}_{stab}$, based on the observation that, for stable event structures $\mathrm{E}, \mathrm{E}_0, \mathrm{E}_1$ the two event structures

$$[\mathrm{E} \oplus \mathrm{E}_0 \to \mathrm{E}_1]$$

and

$$[\mathrm{E} \to [\mathrm{E}_0 \to \mathrm{E}_1]]$$

are the same up to a natural renaming of events.

7.10 Lemma. *Let E, E_0, E_1 be stable event structures. There is a 1-1 correspondence θ between the events of $[E \oplus E_0 \to E_1]$ and $[E \to [E_0 \to E_1]]$ given by*

$$\theta : ((w,x),e) \longleftrightarrow (w,(x,e)),$$

for w, x finite configurations of E, E_0 and event e of E_1, such that θ gives an isomorphism of event structures in the strong sense that

$$a \#_p a' \Leftrightarrow \theta(a) \#_f \theta(a')$$

and

$$X \vdash_p a \Leftrightarrow \theta X \vdash_f \theta(a),$$

where $\#_p, \vdash_p$ are the conflict and entailment relations of $[E_0 \oplus E_1 \to E_2]$ and $\#_f, \vdash_f$ are the relations of $[E_0 \to [E_1 \to E_2]]$.

Certainly the category of stable event structures with stable functions has products including the null event structure as terminal object. The above results yield a natural 1-1 correspondence between morphisms $\mathrm{E}_0 \oplus \mathrm{E}_1 \to \mathrm{E}_2$ and $\mathrm{E}_0 \to [\mathrm{E}_1 \to \mathrm{E}_2]$ and so show the category is cartesian closed [Mac. p.95-96]. We show the exponentiation more explicitly.

7.11 Theorem. *The category $\mathbf{E}_{stab}$ is cartesian closed. It has products as shown and an exponentiation of two stable event structures E_0 and E_1 has the form $[E_0 \to E_1]$, ap where $ap : [E_0 \to E_1] \oplus E_0 \to E_1$ is given by*

$$ap(f, x) = (\phi f)(x)$$

for $f \in \mathcal{F}([E_0 \to E_1])$ and $x \in \mathcal{F}(E_0)$.
(We have identified (f, x) with the corresponding configuration of the disjoint product.)

In the traditional function space used in denotational semantics the functions in the function space $[D \to E]$, where D and E are domains are ordered pointwise, *i.e.* two continuous functions f, g are ordered by

$$f \sqsubseteq g \Leftrightarrow \forall d \in D.\ f(d) \sqsubseteq g(d).$$

This ordering is called the *extensional* order. The inclusion order on the configurations of $[\mathrm{E}_0 \to \mathrm{E}_1]$ induces another order on stable, continuous functions $(\mathcal{F}(\mathrm{E}_0), \subseteq) \to (\mathcal{F}(\mathrm{E}_1), \subseteq)$ which we have seen can be expessed as

$$f \leq g \Leftrightarrow \mu f \subseteq \mu g.$$

This order is called the *stable* order (a name due to Berry). We give an example.

7.12 Example. The two point domain O consisting of $\perp \sqsubseteq T$ can be represented as the the configurations of the obvious event structure with a single event $\bullet$, so $\perp = \emptyset$ and $T = \{\bullet\}$. All the monotonic functions $O \to O$ are stable and continuous. Ordered extensionally they are

$$(\lambda x.\perp) \sqsubseteq (\lambda x.\ x = T \to T|\perp) \sqsubseteq (\lambda x.T)$$

while according to the stable ordering we only have

$$(\lambda x.\perp) \leq (\lambda x.\ x = T \to T|\perp) \quad \text{and} \quad (\lambda x.\perp) \leq (\lambda x.T),$$

because $(\lambda x.\ x = T \to T|\perp) \not\leq (\lambda x.T)$. For two functions to be in the stable order it is not only necessary that they are ordered extensionally but also that if they both output a value for common input then they do so for the same minimal value.

As an example we indicate how the category can be used to give a model for a λ-calculus with atoms.

7.13 Example. We can use the sum construction on event structures (it extends to a functor) and a constant event structure $\mathbf{A}$ of atomic events to define an operation

$$\mathrm{E} \mapsto \mathbf{A} + [\mathrm{E} \to \mathrm{E}].$$

This operation is $\trianglelefteq$-continuous, being the composition of continuous things, and so has a least fixed point which can serve as a model for the λ-calculus with atoms following standard lines.

Recent work of Girard has pointed the way to another appplication for the category of event structures with stable functions, or the equivalent category of dI-domains. In [G], Girard works with a full subcategory of ours with objects called *coherent spaces* and shows how they give a model to his System F, the polymorphic λ-calculus. From the point of view of denotational semantics, coherent spaces are too restrictive because they are not closed under the useful operations of lifting (prefixing) or separated sum. However Girard's ideas can be extended to our category (see [CGW]) which supports such constructions as well as polymorphism. (It should also be mentioned that Girard's ideas also extend, though less directly, to the more traditional category of Scott domains with continuous functions—see [CGW1].)

Another possible use of stable event structures is as a model of Girard's "linear logic" [G1] though this is not yet widely understood . By restricting the category of stable event structures to stable functions which are also

additive (*i.e.* preserve all least uper bounds when they exist) we obtain the subcategory with so-called *linear maps*. This category forms a model for Girard's intuitionistic linear logic (see [YL]) with coherent spaces and linear maps, a model for classical linear logic, as a reflective subcategory. Although these categories are not cartesian closed, they are monoidal closed, a property which means there is a function space but with respect to a bifunctor which is not necessarily a categorical product. Monoidal closed categories have arisen in another context, in the work of Meseguer and Montanari on categories of Petri nets, in generalising the partially synchronous morphisms on nets (see [MM]). The mathematical structure they uncover, as with the monoidal closed categories of event structures, holds great promise, I think, though, at present I don't know how to use it; whereas the partially synchronous morphisms yield familiar and intuitively understandable constructions, underlying working languages like Occam, the linear morphisms give constructions not so familiar in Computer Science. The tie-in with linear logic may be fruitful. While on the topic, I'll mention a that there is a right adjoint to the inclusion functor from stable event structures with partially synchronous morphisms (ignoring labels) and that of stable event structures with finitary linear maps (linear maps preserving finite elements); the right adjoint makes new events out of subsets of the original event structure. Whether this provides any reduction of a linear logic to another more basic one, along the lines of Girard's reduction of intuitionistic to linear logic, I don't know.

8. Further work.

There are two holes in these notes. One is that left by omitting any discussion of logic and proof systems on event structures, and the other is the absence of any treatment of operational semantics and its relation to the event structure models of processes. Some work has been done in both these areas but it is patchy, at least as I perceive it.

A representation of certain kinds of domains was first carried out by G.Kahn and G.Plotkin in their attempt to define a general notion of sequential function (see [KP]). Their work on sequentiality was made higher order by G.Berry and P-L.Curien (see [BC], [C]) though at the expense of moving to algorithms rather than functions. Berry's thesis [Be] is rich in results including the beginnings of an operational characterisation of the stable order between stable functions. As for event structure models of parallel processes and attempts to justify them in operational or observational terms we mention [BoCa], [Ca], [EN], [MDN].

A compositional proof system for Petri nets with categorical combinators of the sort seen here and a recursively defined Hennessy-Milner logic is presented in [W2]. Its generalisation to event structures has proved elusive so far—the thesis [Z] will indicate some of the difficulties. For examples of logics on prime event structures se [TL] and [P].

Appendix: Multisets and multirelations.

Let X and Y be sets. A *multirelation* from X to Y is a function $\alpha : Y \times X \to \omega$, with *entries* $\alpha_{y,x}$ which are nonnegative numbers; we indicate such a multirelation by writing $\alpha : X \to_m Y$. Multirelations are composed as matrices: Let $\alpha : X \to_m Y$ and $\beta : Y \to_m Z$. Define their *composition* $\beta\alpha : X \to_m Z$ to be given by

$$(\beta\alpha)_{z,x} = \Sigma_{y \in Y} \beta_{z,y} \cdot \alpha_{y,x},$$

which exists provided no infinite sums of nonzero integers arise in this way.

A multiset over a set X is a multirelation $\alpha : \mathbf{1} \to_m X$ from a distinguished one-element set $\mathbf{1}$. We write $\alpha \in m(X)$ to mean α is a multiset over X, and write the entry of the multiset α at $x \in X$ as α_x.

Regarded as matrices we can form entrywise sums and scalar products of multirelations, and provided the results never go negative we can subtract one multirelation from another. We write $\alpha + \beta$ and $\alpha - \beta$ for the sum and difference of $\alpha, \beta : X \to_m Y$. Similarly, we can write $\alpha \leq \beta$, for $\alpha, \beta : X \to_m Y$, when $\alpha_{x,y} \leq \beta_{x,y}$ for all $x \in X, y \in Y$.

We identify subsets of a set X with those multisets over it whose entries never exceed 1. In particular, the *null* multiset $\mathbf{0}$ of X for which every entry is 0 corresponds to the empty set. For $x \in X$, we shall write simply x for the singleton multiset which has entry 1 at x and 0 elsewhere. We shall identify *relations* between a set X and a set Y with those multirelations $\theta : X \to_m Y$ for which all entries are at most 1. In particular, a partial function $f : X \to_* Y$, between sets X and Y, is identified with a multirelation $f : X \to_m Y$ in which $f_{y,x} \leq 1$, for all $x \in X, y \in Y$, and for any $x \in X$ there is at most one $y \in Y$ so $f_{y,x} = 1$; the fact that an application $f(x)$ of a partial function f to an element x is undefined corresponds to the application fx of the multirelation f to the singleton multiset x being $\mathbf{0}$.

Acknowledgements: Thanks to Mogens Nielsen for helpful comments.

References

[Be] Berry, G., Modèles complètement adéquats et stables des lambda-calculs typés. Thèse de Doctorat d'Etat, Université de Paris VII (1979).

[B] Bednarczyk, M.A., Categories of asynchronous systems. PhD in Comp Sc, University of Sussex, report no.1/88 (1988).

[BC] Berry, G., and Curien, P-L., Sequential algorithms on concrete data structures. TCS vol 20 (1982).

[BoCa]Boudol,G., and Castellani,I., On the semantics of concurrency: partial orders and transition systems. Springer Lec Notes in Comp Sc vol 249 (1987).

[C]Curien, P-L., Categorical combinators, sequential algorithms and functional programming. Research notes in theoretical comp. sc., Pitman, London (1986).

[Ca] Castellani, I., Permutations of transitions. This volume.

[EN] Engberg,U., Nielsen,M., and Larsen,K.S., Fully abstract models of a language with refinement. This volume.

[CGW] Coquand, T., Gunter, C., and Winskel, G., Polymorphism and domain equations. In the proc of Third Workshop on the Mathematical Foundations of Programming Language Semantics, New Orleans, LA 1987.

[CGW1] Coquand, T., Gunter, C., and Winskel, G., Domain theoretic models of polymorphism. To appear in Information and Computation, 1987.

[G] Girard, J-Y., The system F of variable types, fifteen years later. TCS vol.45 (1986).

[G1] Girard, J-Y., Linear logic. TCS 1987.

[KP] Kahn, G., and Plotkin, G., Domaines Concrètes. Rapport IRIA Laboria No. 336 (1978).

[YL] Lafont, Y., Linear logic and lazy computation. Springer Lec Notes in Comp Sc vol 249 (1987).

[Lam] Lamport, L., Time clocks and the ordering of events in a distributed system. CACM 21, (1978).

[Mac] Maclane, S., Categories for the Working Mathematician. Graduate Texts in Mathematics, Springer (1971).

[Maz] Mazurkiewicz, A., Traces. This volume.

[Mil]Milner, R., A Calculus of Communicating Systems. Springer Lecture Notes in Comp. Sc. vol. 92 (1980).

[MM] Meseguer, J., and Montanari, U., Petri nets are monoids: a new algebraic foundation for net theory. Proc of LICS, Computer Society Press (1988).

[MDN] Degano, P., De Nicola,R. and Montanari, U., On the consistency of "truly concurrent" operational and denotational semantics. Proc of LICS, Computer Society Press (1988).

[NPW] Nielsen, M., Plotkin, G., Winskel, G., Petri nets, Event structures and Domains, part 1 . Theoretical Computer Science, vol. 13 (1981).

[P] Penczek,W., The temporal logic of event structures. Report 616,

Inst of Comp Sc, Polish Academy of Science, Warsaw, 1987.

[Rem] Rem, M., Partially ordered computations, with an application to VLSI design. In "Foundations of Computer Science IV, part 2", Mathemtical Centre, Amsterdam (1983).

[TL] Lodaya,K., and Thiagarajan, P.S., A modal logic for a subclass of event structures. Proc of ICALP 1987 published in Springer Lecture Notes in C.S. (1987).

[W] Winskel, G., Events in Computation. Ph.D. thesis, available as a technical report, Comp. Sc. Dept., University of Edinburgh (1980).

[W1]Winskel, G., Event structures. Invited lectures for the Advanced Course on Petri nets, September 1986. Appears as a report of the Computer Laboratory, University of Cambridge, 1986, and in the proceedings of the school, published in Springer Lecture Notes in C.S., vol.255 (1987).

[W2] Winskel, G., A category of labelled nets and compositional proof system. Proc of LICS, Computer Society Press (1988).

[Z] Zhang, G.Q., Logic and semantics in computation. PhD in progress, Computer Laboratory, Cambridge University.