MODELS FOR CONCURRENCY

Glynn Winskel   Mogens Nielsen
Computer Science Department, Aarhus University, Denmark

Contents

1 Introduction

The purpose of this chapter is to provide a survey of the fundamental models for distributed computations used and studied within theoretical computer science. Such models have the nature of mathematical formalisms in which to describe and reason about the behaviour of distributed computational systems. Their purpose is to provide an understanding of systems and their behaviour in theory, and to contribute to methods of design and analysis in practice.

In the rich theory of sequential computational systems, several mathematical models have been studied in depth, *e.g.* Turing machines, lambda calculus, Post systems, Markov systems, random access machines, *etc.* A main result of this theory is that the formalisms are all equivalent, in the sense that their behaviours in terms of input-output functions are the same.

However, in reality, few computational systems are sequential. On all levels, from a small chip to a world-wide network, computational behaviours are often distributed, in the sense that they may be seen as spatially separated activities accomplishing a joint task. Many such systems are not meant to terminate, and hence it makes little sense to talk about their behaviours in terms of traditional input-output functions. Rather, we are interested in the behaviour of such systems in terms of the often complex patterns of stimuli/response relationships varying over time. For this reason such systems are often referred to as *reactive systems*.

In the study of reactive systems, we are forced to take a different, less abstract, view of behaviours than the traditional one equating behaviour with an input-output function. A notion of behaviour is needed which expresses the patterns of actions which a system can perform, so as to capture such aspects as deadlock, mutual exclusion, starvation, *etc.*

One may see such models as providing a foundation for the development of all other theoretical and practical research areas on distributed computing. To give some examples, the models are used to provide the semantics of process description languages, and hence the basis of the many behavioural equivalences studied in the literature on process calculi. They are used to give the formal definition of specification logics, and hence underpin the work on verification of systems with respect to such specifications. And given this, they are at the heart of the development of automated tools for reasoning about distributed systems.

Numerous models have been suggested and studied over the past 10-15 years. Here we shall not attempt to present a complete survey. Rather, we have chosen to present in fair detail a few key models.

Common to all the models we consider, is that they rest on the central idea of atomic actions, over which the behaviour of a system is defined. The models differ mainly with respect to what behavioural features of systems are represented. Some models are more abstract than others, and this fact is often used in informal classifications of the models with respect to expressibility. One of our aims is to present principal representatives of models, covering the landscape from the most abstract to the most concrete, and to formalise the nature of their relationships by explicitly representing the steps of abstraction that are involved in moving between them. In other words we would like to set the scene for a formal classification of models.

Let us be more specific. Imagine a very simple distributed computational system consisting of three individual components, each performing some independent computations, involving one (Sender) occasionally sending messages to another (Receiver) via the third (Medium):

$$\text{Sender} \quad \longleftrightarrow \quad \text{Medium} \quad \longleftrightarrow \quad \text{Receiver}$$

Imagine modelling the behaviour of this system in terms of some atomic actions of the individual components, and the two actions of delivering a message from Sender to Medium, and passing on a message from Medium to Receiver. Obviously, having fixed such a set of atomic actions, we have also fixed a particular physical level at which to model our system.

One main distinction in the classification of models is that between interleaving and noninterleaving models. The main characteristic of an interleaving model is that it abstracts away from the fact that our system is actually composed of three independently computing agents, and models the behaviour in terms of purely sequential patterns of actions. Formally, the behaviour of our system will be expressed in terms of the nondeterministic merging, or interleaving, of the sequential behaviours of the three components. Prominent examples of such models are transition systems [42], synchronisation trees [54], acceptance trees [35], and Hoare traces [33].

It is important to realise that in many situations abstraction like this is exactly what is wanted, and it has been demonstrated in the references above that many interesting and important properties of distributed systems may be expressed and proved based on interleaving models. The whole point of abstraction is, of course, to ignore aspects of the system which are irrelevant for the features we would like to reason about.

However, there may be situations in which it is important to keep the information that our system is composed of the three independently computing components, a possibility offered by the so-called noninterleaving models, with Petri nets [2], event structures [97], and Mazurkiewicz traces [62] as prime examples. One such situation is that where some behavioural properties (typically liveness properties) rest on the fact that each component is a separate physical entity independently making its own computational progress. Dealing with such properties in interleaving models is often handled by a specific naming of the actions belonging to the components combined with logical assertions expressing progress assumptions for the system under study, *i.e.* handled outside the model in an *ad hoc* fashion.

Another issue is how models deal with the concept of nondeterminism in computations, distinguishing between so-called linear-time and branching-time models.

Imagine that in our system the component Medium is erroneous, in the sense that delivering a message from the Sender may leave Medium in either a normal state, having accepted the message and ready for another delivery or a passing of a message to Receiver, or a faulty state insisting on another delivery. A linear-time model will abstract away from this possibility of a suspended behaviour of the process Medium (and hence from some possibilities of deadlock). These models typically express the full nondeterministic behaviour of a system in terms of its set of possible (determinate) "runs" (or computation paths). Major examples of the structures used to model runs are Hoare traces, Mazurkiewicz traces and Pratt's pomsets.

As indicated, in many situations a more detailed representation of when nondeterministic choices are made during a computation is necessary to reflect absence of deadlocks and other safety properties of systems. This is possible to various degrees in branching time models like synchronisation and acceptance trees, Petri nets, and event structures. Of course, the treatment of nondeterminism is particularly important for the interleaving models, where parallel activities are also expressed in terms of nondeterminism.

Finally, yet a third distinction is made between those models allowing an explicit representation of the (possibly repeating) states in a system, and models abstracting away from such information, which focus instead on the behaviour in terms of patterns of occurrences of actions over time. Prime examples of the first type are transition systems and Petri nets, and of the second type, trees, event structures and traces.

Thus the seemingly confusing world of models for concurrency can be structured according to a classification based on the expressiveness of the various models. In following through this programme, category theory is a convenient language for formalising the relationships between models.

To give an idea of the role categories play, let's focus attention on transition systems as a model of parallel computation. A transition system consists of a set of states with labelled transitions between them. Assume the transition system has a distinguished initial state so that it can be presented by

$$(S, i, L, \mathit{Tran})$$

where S is a set of states with initial state i, L is a set of labels and the transitions elements of $\mathit{Tran} \subseteq S \times L \times S$; a transition (s, a, s') is generally written as $s \xrightarrow{a} s'$. It then models a process whose transitions represent the process's atomic actions while the labels are action names; starting from the initial state, it traces out a computation path as transitions occur consecutively.

Processes often arise in relationship to other processes. For example, one process may refine another, or perhaps one process is a component of another. The corresponding relationships between behaviours are often expressed as morphisms between transition systems. For several models, there is some choice in how to define appropriate morphisms—it depends on the extent of the relationship between processes we wish to express. But here, we have an eye to languages like CCS, where communication is based on the synchronisation of atomic actions. From this viewpoint, we get a useful class of morphisms, sufficient to relate the behaviour of processes and their subcomponents, by taking a morphism from one transition system T to another T' to be a pair (σ, λ), in which

- σ is a function from the states of T to those of T' that sends the initial state of T to that of T',
- λ is a *partial* function from the labels of T to those of T' such that for any transition $s \xrightarrow{a} a'$ of T if $\lambda(a)$ is defined, then $\sigma(s) \xrightarrow{\lambda(a)} \sigma(s')$ is a transition of T'; otherwise, if $\lambda(a)$ is undefined, then $\sigma(s) = \sigma(s')$.

Morphisms respect a choice of granularity for actions in the sense that an action may only be sent to at most one action, and not to a computation consisting of several actions. By taking λ to be a *partial* function on labels, we in particular accommodate the fact that projecting from a parallel composition of processes (*e.g.* in CCS) to a component may not only change action names, but also allow some actions to vanish if they do not correspond to those of the component, in which case their occurrence has no effect on the state of the component.

This definition of morphism is sufficient to express the relationship between a constructed process and its components as morphisms, at least within a language like CCS. But conversely the choice of morphisms also produces constructions. This is because transition systems and their morphisms form a category, and universal constructions (including limits and colimits) of a category are determined uniquely to within isomorphism, once they exist. In fact the universal constructions of the category of transition systems form the basis of a process description language. It is a little richer than that of CCS and CSP in the sense that their operations are straightforwardly definable within it.

When we consider other models as categories the same universal constructions yield sensible interpretations of the process-language constructs. Without categories this unity is lost; indeed, the denotations of parallel compositions, often nontrivial to define, have been invented in an *ad hoc* fashion for most of the models we present.

Categorical notions also come into play in relating different models. Another model, synchronisation trees, arises by ignoring repetitive behaviour. We can identify synchronisation trees with special transition systems (those with no loops, no distinct transitions *to* the same state, in which all states are reachable). Synchronisation trees inherit morphisms from transition systems, and themselves form a category. The inclusion of synchronisation trees in transition systems is a functor. But more, the inclusion functor is part of an adjunction; the inclusion functor (the left adjoint) is accompanied, in a uniquely-determined way, by a functor (the right adjoint) unfolding transition systems to synchronisation trees. A further step of abstraction, this time ignoring the branching of computation paths, takes us to languages as models of processes. A process is represented by the set of strings of actions it can perform. Languages can be identified with certain kinds of synchronisation trees and again this inclusion is part of an adjunction.

$$\text{Languages} \hookrightarrow \begin{array}{l}\text{Synchronisation}\\ \text{trees}\end{array} \hookrightarrow \begin{array}{l}\text{Transition}\\ \text{systems}\end{array}$$

As parts of adjunctions the functors enjoy preservation properties, which coupled with the understanding of process operations as universal constructions, are useful in relating different semantics.

Here we have discussed just the three simplest models, but the same general approach applies to other models. The main idea is that each model will be equipped with a notion of morphism, making it into a category in which the operations of process calculi are universal constructions. The morphisms will preserve behaviour, at the same time respecting a choice of granularity of actions in the description of processes. One role of the morphisms is to relate the behaviour of a construction on processes to that of its components. As we shall see, it turns out that certain kinds of adjunctions (reflections and coreflections[1]) provide a way to express that one model is embedded in (is more abstract than) another, even when the two models are expressed in very different mathematical terms. One adjoint will say how to embed the more abstract model in the other, the other will abstract away from some aspect of the representation, in the same manner as has been described above. The adjunctions not only provide an aid in the understanding of the different models and their relationships, but are also a vehicle for the transfer of techniques from one model to another. In this chapter we concentrate on the role of models in giving a formal semantics to process description languages. The understanding of their operations as universal constructions guides us away from *ad hoc* definitions. And importantly, we can use the preservation properties of adjoints to show how a semantics in one model translates to a semantics in another.

In summary, our goal is to survey a few, fundamental models for concurrency, and exploit category theory as a language for describing their structure and their relationships.[2]

---

[1] A *reflection* is an adjunction in which the right adjoint is full and faithful, a *coreflection* one where the left adjoint is full and faithful

[2] A knowledge of basic category theory, up to an acquaintance with the notion of adjunction, is sufficient for the whole chapter. However a light aquaintance, and some goodwill, should suffice, at least for the earlier parts. Good introductory references are [69] and [4], while [50] remains the classic text.

2 Transition systems

Transition systems are a commonly used and understood model of computation. They provide the basic operational semantics for Milner's Calculus of Communicating Systems (CCS) and often underlie other approaches, such as that of Hoare's Communicating Sequential Processes (CSP). The constructions on transition systems used in such methods can frequently be seen as universal in a category of transition systems, where the morphisms can be understood as expressing the partial simulation (or refinement) of one process by another. By "abstract nonsense" the universal properties will characterise the constructions to within isomorphism. More strikingly, the same universal properties will apply in the case of other models like Petri nets or event structures, which are seemingly very different in nature.

2.1 A category of transition systems

Transition systems consist of a set of states, with an initial state, together with transitions between states which are labelled to specify the kind of events they represent.

Definition: A *transition system* is a structure

$$(S, i, L, \mathit{Tran})$$

where

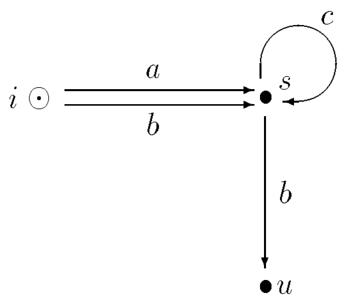
- S is a set of *states* with *initial state* i,
- L is a set of *labels*, and
- $\mathit{Tran} \subseteq S \times L \times S$ is the *transition relation*.

This definition narrows attention to transition systems, which are *extensional*: no two distinct transitions with the same label have the same pre and post states.

Notation: Let (S, i, L, Tran) be a transition system. We write

$$s \xrightarrow{a} s'$$

to indicate that $(s, a, s') \in \mathit{Tran}$.This notation lends itself to the familiar graphical notation for transition systems. For example,



represents a transition system which at the initial state i (encircled to distinguish it) can perform either an a or a b transition to enter the state s at which it can repeatedly perform a c transition or a b transition to enter state u.

We occasionally write

$$s \stackrel{a}{\not\longrightarrow}$$

to mean there is no transition (s, a, s'). It is sometimes convenient to extend the arc-notation to strings of labels and write

$$s \xrightarrow{v} s',$$

when $v = a_1 a_2 \cdots a_n$ is a, possibly empty, string of labels in L, to mean

$$s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n,$$

for some states $s_1, \ldots, s_n$. A state s is said to be *reachable* when $i \xrightarrow{v} s$ for some string v.

Definition: Say a transition system $T = (S, i, L, \mathit{Tran})$ is *reachable* iff every state in S is reachable from i and for every label a there is a transition $(s, a, s') \in \mathit{Tran}$. Say T is *acyclic* iff, for all strings of labels v, if $s \xrightarrow{v} s$ then v is empty.

It is convenient to introduce *idle* transitions, associated with any state. This has to do with our representation of partial functions, explained in Appendix A. We view a partial function from a set L to a set L' as a (total) function $\lambda : L \to L \cup \{*\}$, where $*$ is a distinguished element standing for "undefined". This representation is reflected in our notation $\lambda : L \to_* L'$ for a partial function λ from L to L'. It assumes that $*$ does not appear in the sets L and L', and more generally we shall assume that the reserved element $*$ does not appear in any of the sets appearing in our constructions.

Definition: Let $T = (S, i, L, \mathit{Tran})$ be a transition system. An *idle transition* of T typically consists of $(s, *, s)$, where $s \in S$. Define

$$\mathit{Tran}_* = \mathit{Tran} \cup \{(s, *, s) \mid s \in S\}.$$

Idle transitions help simplify the definition of morphisms between transition systems. Morphisms on transitions systems have already been discussed in the Introduction. There, a morphism $T \to T'$ between transition systems was presented as consisting of a pair, one component σ being a function on states, preserving initial states, and the other a partial function λ on labels with the property that together they send a transition of T to a transition of T', whenever this makes sense. More precisely, if (s, a, s') is a transition of T then $(\sigma(s), \lambda(a), \sigma(s'))$ is a transition of T' provided $\lambda(a)$ is defined; otherwise, in the case where $\lambda(a)$ is undefined, it is insisted that the two states $\sigma(s)$ and $\sigma(s')$ are equal. With the device of idle transitions and the particular representation of partial functions, the same effect is achieved with the following definition:

Definition: Let

$$T_0 = (S_0, i_0, L_0, \mathit{Tran}_0) \text{ and}$$
$$T_1 = (S_1, i_1, L_1, \mathit{Tran}_1)$$

be transition systems. A *morphism* $f : T_0 \to T_1$ is a pair $f = (\sigma, \lambda)$ where

- $\sigma : S_0 \to S_1$
- $\lambda : L_0 \to_* L_1$ are such that $\sigma(i_0) = i_1$ and

$$(s, a, s') \in \mathit{Tran}_0 \Rightarrow (\sigma(s), \lambda(a), \sigma(s')) \in \mathit{Tran}_{1*}.$$

The intention behind the definition of morphism is that the effect of a transition with label a in T_0 leads to inaction in T_1 precisely when $\lambda(a)$ is undefined. In our definition of morphism, idle transitions represent this inaction, so we avoid the fuss of considering whether or not $\lambda(a)$ is defined. With the introduction of idle transitions, morphisms on transition systems can be described as preserving transitions and the initial state. It is stressed that an idle transition $(s, *, s)$ represents inaction, and is to be distinguished from the action expressed by a transition (s, a, s') for a label a.

Morphisms preserve initial states and transitions and so clearly preserve reachable states:

Proposition 1 *Let $(\sigma, \lambda) : T_0 \to T_1$ be a morphism of transition systems. Then if s is a reachable state of T_0 then $\sigma(s)$ is a reachable state of T_1.*

Transition systems and their morphisms form a category which will be the first important category in our study:

Proposition 2 *Taking*

- *the class of objects to be transition systems,*
- *the class of morphisms to be those of transition systems,*

defines a category, where

- *the composition of two morphisms* $f = (\sigma, \lambda) : T_0 \to T_1$ *and* $g = (\sigma', \lambda') : T_1 \to T_2$ *is* $g \circ f = (\sigma' \circ \sigma, \lambda' \circ \lambda) : T_0 \to T_2$*—here composition on the left of a pair is that of total functions while that on the right is of partial functions, and*

- *the identity morphism for a transition system* T *has the form* $(1_S, 1_L)$*, where* 1_S *is the identity function on states* S *and* 1_L *is the identity function on the labelling set* L *of* T*.*

Proof: It is easily checked that composition is associative and has the identities claimed. $\square$

Definition: Denote by $\mathbf{T}$ the category of labelled transition systems given by the last proposition.

2.2 Constructions on transition systems

Transition systems are used in many areas. We focus on their use in modelling process calculi. The constructions used there can be understood as universal constructions in the category of transition systems. The point is not to explain the familiar in terms of the unfamiliar, but rather to find characterisations of sufficient generality that they apply to the other models as well. As we will see, the category of transition systems is rich in categorical constructions which furnish the basic combinators for a language of parallel processes.

2.2.1 Restriction

Restriction is an important operation on processes. For example, in Milner's CCS, labels are used to distinguish between input and output to channels, connected to processes at ports, and internal events. The effect of hiding all but a specified set of ports of a process, so that communication can no longer take place at the hidden ports, is to restrict the original behaviour of the process to transitions which do not occur at the hidden ports. Given a transition system and a subset of its labelling set, the operation of restriction removes all transitions whose labels are not in that set:

Definition: Let $T' = (S, i, L', \mathit{Tran}')$ be a transition system. Assume $L \subseteq L'$ and let $\lambda : L \hookrightarrow L'$ be the associated inclusion morphism, taking a in L to a in L'. Define the *restriction* $T' \upharpoonright \lambda$ to be the transition system (S, i, L, Tran) with

$$\mathit{Tran} = \{(s, a, t) \in \mathit{Tran}' \mid a \in L\}.$$

Restriction is a construction which depends on labelling sets and functions between them. Seeing it as a categorical construction involves dealing explicitly with functions on labelling sets and borrowing a fundamental idea from fibred category theory. We observe that there is a functor $p : \mathbf{T} \to \mathbf{Set}_*$, to the category of sets with partial functions, which sends a morphism of transition systems $(\sigma, \lambda) : T \to T'$ between transition systems T over L and T' over L' to the partial function $\lambda : L \to_* L'$. Associated with a restriction $T' \upharpoonright L$ is a morphism $f : T' \upharpoonright L \to T'$, given by $f = (1_S, \lambda)$ where λ is the inclusion map $\lambda : L \hookrightarrow L'$. In fact the morphism f is essentially an "inclusion" of the restricted into the original transition system. The morphism f associated with the restriction has the universal property that:

> For any $g : T \to T'$ a morphism in $\mathbf{T}$ such that $p(g) = \lambda$ there is a unique morphism $h : T \to T' \upharpoonright L$ such that $p(h) = 1_L$ and $f \circ h = g$. In a diagram:

$$\begin{tikzcd} & T \arrow[d, dashed, "h"'] \arrow[dr, "g"] & \\ \mathbf{T} \arrow[dd, "p"'] & T' \upharpoonright L \arrow[r, "f"] & T' \\ & & \\ \mathbf{Set}_* & L \arrow[r, "\lambda"] & L' \end{tikzcd}$$

This says that the "inclusion" morphisms associated with restrictions are *cartesian*; the morphism f is said to be a *cartesian lifting* of λ with respect to T'. In fact, they are *strong* cartesian—see Appendix B.

Proposition 3 *Let $\lambda : L \to_* L'$ be an inclusion. Let T' be a labelled transition system, with states S. There is a morphism $f : T' \upharpoonright L \to T'$, given by $f = (1_S, \lambda)$. It is (strong) cartesian.*

Because an inclusion $\lambda : L \hookrightarrow L'$ has cartesian liftings for any T' with labelling set L', restriction automatically extends to a functor from transition systems with labelling set L' to those with labelling set L. To state this more fully, note first that a labelling set L is associated with a subcategory of transition systems $p^{-1}(L)$, called the *fibre* over L, consisting of objects those transition systems T for which $p(T) = L$ (*i.e.* whose labelling set is L) and morphisms h for which $p(h) = 1_L$ (*i.e.* which preserve labels). An explicit choice of cartesian lifting for each T' in $p^{-1}(L')$ (as is provided by the restriction operation) yields a functor between fibres $p^{-1}(L') \to p^{-1}(L)$—the functor's action on morphisms coming from the universal property of cartesian liftings.

Remark: In fact there are (strong) cartesian liftings for any $\lambda : L \to_* L'$ and any T' with labelling set L', and the functor $p : \mathbf{T} \to \mathbf{Set}_*$ is a fibration.

2.2.2 Relabelling

In CCS, one can make copies of a process by renaming its port names. This is associated with the operation of relabelling the transitions in the transition system representing its behaviour. When $\lambda : L \rightarrow_* L'$ is total, the relabelling construction takes a transition system T with labelling set L to $T\{\lambda\}$, the same underlying transition system but relabelled according to λ.

Definition: Let $T = (S, i, L, \mathit{Tran})$ be a transition system. Let $\lambda : L \rightarrow L'$ be a total function. Define the *relabelling* $T\{\lambda\}$ to be the transition system $(S, i, L', \mathit{Tran}')$ where

$$\mathit{Tran}' = \{(s, \lambda(a), s') \mid (s, a, s') \in \mathit{Tran}\}.$$

The operation of relabelling is associated with a construction dual to that of cartesian lifting, that of forming a *cocartesian lifting*. Letting the transition system T have states S, there is a morphism $f = (1_S, \lambda) : T \rightarrow T\{\lambda\}$. Such a morphism is a cocartesian lifting of λ in the sense that:

> For any $g : T \rightarrow T'$ a morphism in $\mathbf{T}$ such that $p(g) = \lambda$ there is a unique morphism $h : T\{\lambda\} \rightarrow T'$ such that $p(h) = 1_{L'}$ and $h \circ f = g$. In a diagram:

$$\begin{tikzcd}
 & & T' \\
\mathbf{T} \arrow[d, "p"'] & T \arrow[r, "f"] \arrow[ur, "g"] & T\{\lambda\} \arrow[u, dashed, "h"'] \\
\mathbf{Set}_* & L \arrow[r, "\lambda"] & L'
\end{tikzcd}$$

Proposition 4 *Let $\lambda : L \rightarrow L'$ be a total function. Let T be a labelled transition system, with states S. There is a morphism $f : T \rightarrow T\{\lambda\}$, given by $f = (1_S, \lambda)$ which is (strong) cocartesian—see Appendix B.*

Relabelling extends to a functor $p^{-1}(L) \rightarrow p^{-1}(L')$, where $\lambda : L \rightarrow L'$ is a total function.

Remark: The relabelling construction can also be defined more generally when λ is partial. In fact there are (strong) cocartesian liftings for any $\lambda : L \rightarrow_* L'$ and any T with labelling set L, and the functor $p : \mathbf{T} \rightarrow \mathbf{Set}_*$ is a cofibration. Being a fibration too, this makes p a bifibration.

2.2.3 Product

Parallel compositions are central operations in process calculi; they set processes in communication with each other. Communication is via actions of mutual synchronisation, possibly with the exchange of values. Precisely how actions synchronise with each other varies enormously from one language to another, but for example in CCS and Occam processes are imagined to communicate over channels linking their ports. In these languages, an input action to a channel from one process can combine with an output action to the same channel from the other to form an action of synchronisation. The languages also allow for processes in a parallel composition to reserve the possibility of communicating with a, yet undetermined, process in the environment of both.

Parallel compositions in general can be derived, with restriction and relabelling, from a product operation on transition systems. In itself the product operation is a special kind of parallel composition in which all conceivable synchronisations are allowed.

Definition: Assume transition systems $T_0 = (S_0, i_0, L_0, \mathit{Tran}_0)$ and $T_1 = (S_1, i_1, L_1, \mathit{Tran}_1)$. Their *product* $T_0 \times T_1$ is (S, i, L, Tran) where

- $S = S_0 \times S_1$, with $i = (i_0, i_1)$, and projections $\rho_0 : S_0 \times S_1 \to S_0$, $\rho_1 : S_0 \times S_1 \to S_1$
- $L = L_0 \times_* L_1 =$
  $\{(a, *) \mid a \in L_0\} \cup \{(*, b) \mid b \in L_1\} \cup \{(a, b) \mid a \in L_0, b \in L_1\}$,
  with projections π_0, π_1, and
- $(s, a, s') \in \mathit{Tran}_* \Leftrightarrow$
  $(\rho_0(s), \pi_0(a), \rho_0(s')) \in \mathit{Tran}_{0*}$ & $(\rho_1(s), \pi_1(a), \rho_1(s')) \in \mathit{Tran}_{1*}$.

Define $\Pi_0 = (\rho_0, \pi_0)$ and $\Pi_1 = (\rho_1, \pi_1)$.

Example: Let T_0 and T_1 be the following transition systems

where T_0 has $\{a\}$ and T_1 has $\{b\}$ as labelling set. The product of these labelling sets is

$$\{a\} \times_* \{b\} = \{(a, *), (a, b), (*, b)\}$$

with projections λ_0 onto the first coordinate and λ_1 onto the second. Thus $\lambda_0(a, *) = \lambda_0(a, b) = a$ and $\lambda_0(*, b) = *$. Their product takes the form:

$$\begin{tikzcd}
& \bullet & \\
\bullet \arrow[ur, "{(*,b)}"] & & \bullet \arrow[ul, "{(a,*)}"'] \\
& \odot \arrow[ul, "{(a,*)}"] \arrow[ur, "{(*,b)}"'] \arrow[uu, "{(a,b)}"] &
\end{tikzcd}$$

Intuitively, transitions with labels of the form (a, b) represent synchronisations between two processes set in parallel, while those labelled $(a, *)$ or $(*, b)$ involve only one process, performing the transition unsynchronised with the other. Clearly, this is far too "generous" a parallel composition to be useful as it stands, allowing as it does all possible synchronisations and absences of synchronisations between two processes. However, a wide range of familiar and useful parallel compositions can be obtained from the product operation by further applications of restriction (to remove unwanted synchronisations) and relabelling (to rename the results of synchronisations).

The product of transition systems T_0, T_1 has projection morphisms $\Pi_0 = (\rho_0, \pi_0) : T_0 \times T_1 \to T_0$ and $\Pi_1 = (\rho_1, \pi_1) : T_0 \times T_1 \to T_1$. They together satisfy the universal property required of a *product in a category*; *viz.* given any morphisms $f_0 : T \to T_0$ and $f_1 : T \to T_1$ from a transition system T there is a unique morphism $h : T \to T_0 \times T_1$ such that $\Pi_0 \circ h = f_0$ and $\Pi_1 \circ h = f_1$:

$$\begin{tikzcd}
T_0 & T_0 \times T_1 \arrow[l, "\Pi_0"'] \arrow[r, "\Pi_1"] & T_1 \\
& T \arrow[ul, "f_2"] \arrow[u, dashed, "h"] \arrow[ur, "f_1"'] &
\end{tikzcd}$$

Proposition 5 *Let T_0 and T_1 be transition systems. The construction $T_0 \times T_1$ above, with projections $\Pi_0 = (\rho_0, \pi_0)$, $\Pi_1 = (\rho_1, \pi_1)$, is a product in the category* **T**. *A state s is reachable in $T_0 \times T_1$ iff $\rho_0(s)$ is reachable in T_0 and $\rho_1(s)$ is reachable in T_1.*

Although we have only considered binary products, all products exist in the category of transition systems. In particular, the empty product is the *nil* transition system

$$nil = (\{i\}, i, \emptyset, \emptyset),$$

consisting of a single initial state i. In this special case the universal property for products amounts to:

for any transition system T, there is a unique morphism $h : T \to nil$, that is, *nil* is a *terminal* object in the category of transition systems. The transition system *nil* is also an *initial* object in the category of transition systems:

for any transition system T, there is a unique morphism $h : nil \to T$.

We remark that the product-machine contruction from automata theory arises as a *fibre product*, *viz.* a product in a fibre. Recall a fibre $p^{-1}(L)$ is a category which consists of the subcategory of transition systems with a common labelling set L, in which the morphisms preserve labels.

2.2.4 Parallel compositions

In the present framework, we do not obtain arbitrary parallel compositions as single universal constructions. Generally, they can be obtained from the product by restriction and relabelling; a parallel composition of T_0 and T_1, with labelling sets L_0, L_1 respectively, is got by first taking their product, to give a transition system $T_0 \times T_1$ with labelling set $L_0 \times_* L_1$, then restricting by taking $(T_0 \times T_1) \upharpoonright S$ for an inclusion $S \subseteq L_0 \times_* L_1$, followed by a relabelling $((T_0 \times T_1) \upharpoonright S)\{r\}$ with respect to a total function $r : S \to L$. In this way, using a combination of product, restriction and relabelling we can represent all conceivable parallel compositions which occur by synchronisation.

In general parallel compositions are derived using a combination of product, restriction and relabelling. We can present the range of associative, commutative parallel compositions based on synchronisation in a uniform way by using *synchronisation algebras*. A synchronisation algebra on a set L of labels (not containing the distinct elements $*, 0$) consists of a binary, commutative, associative operation $\bullet$ on $L \cup \{*, 0\}$ such that

$$a \bullet 0 = 0 \text{ and } (a_0 \bullet a_1 = * \Leftrightarrow a_0 = a_1 = *)$$

for all $a, a_0, a_1 \in L \cup \{*, 0\}$. The role of 0 is to specify those synchronisations which are not allowed whereas the composition $\bullet$ specifies a relabelling. (It may be helpful to look at the example ahead of the synchronisation algebra of CCS.) For a synchronisation algebra on labels L, let $\lambda_0, \lambda_1 : L \times_* L \to_* L$ be the projections on its product in $\mathbf{Set}_*$. The parallel composition of two transition systems T_0, T_1, labelled over L, can be obtained as $((T_0 \times T_1) \upharpoonright D)\{r\}$ where $D \subseteq L \times_* L$ is the inclusion of

$$D = \{a \in L \times_* L \mid \lambda_0(a) \bullet \lambda_1(a) \neq 0\}$$

determined by the 0-element, and $r : D \to L$ is the relabelling, given by

$$r(a) = \lambda_0(a) \bullet \lambda_1(a)$$

for $a \in D$.

We present two synchronisation algebras as examples, in the form of tables—more, including those for value-passing, can be found in [92, 94].

Example: *The synchronisation algebra for pure CCS:* In CCS [55] events are labelled by $a, b, \cdots$ or by their complementary labels $\overline{a}, \overline{b}, \cdots$ or by the label τ. The idea is that only two events bearing complementary labels may synchronise to form a synchronisation event labelled by τ. Events labelled by τ cannot synchronise further; in this sense they are invisible to processes in the environment, though their occurrence may lead to internal changes of state. All labelled events may occur asynchronously. Hence the synchronisation algebra for CCS takes the following form. The resultant parallel composition, of processes p and q say, is represented as $p|q$ in CCS.

$\bullet$	$*$	a	$\overline{a}$	b	$\overline{b}$	$\cdots$	τ	0
$*$	$*$	a	$\overline{a}$	b	$\overline{b}$	$\cdots$	τ	0
a	a	0	τ	0	0	$\cdots$	0	0
$\overline{a}$	$\overline{a}$	τ	0	0	0	$\cdots$	0	0
b	b	0	0	0	τ	$\cdots$	0	0
.	.	.	.	.	.	$\cdots$	.	.

Example: *The synchronisation algebra in TCSP:* In "theoretical" CSP—see [34, 15]—events are labelled by $a, b, \cdots$ or τ. For one of its parallel composition (usually written $\|$) events must "synchronise on" $a, b, \cdots$. In other words non–τ–labelled events cannot occur asynchronously. Rather, an a–labelled event in one component of a parallel composition must synchronise with an a–labelled event from the other component in order to occur; the two events must synchronise to form a synchronisation event again labelled by a. The synchronisation algebra for this parallel composition takes the following form.

$\bullet$	$*$	a	b	$\cdots$	τ	0
$*$	$*$	0	0	$\cdots$	τ	0
a	0	a	0	$\cdots$	0	0
b	0	0	b	$\cdots$	0	0
.	.	.	.	$\cdots$	.	.

2.2.5 Sum

Nondeterministic sums in process calculi allow a process to be defined with the capabilities of two or more processes, so that the process can behave like one of several alternative processes. Which alternative can depend on what communications the environment offers, and in many cases, nondeterministic sum plays an important role like that of the conditional of traditional sequential languages.

With the aim of understanding nondeterministic sums as universal constructions we examine coproducts in the category of transition systems.

Definition: Let $T_0 = (S_0, i_0, L_0, Tran_0)$ and $T_1 = (S_1, i_1, L_1, Tran_1)$ be transition systems. Define $T_0 + T_1$ to be $(S, i, L, Tran)$ where

- $S = (S_0 \times \{i_1\}) \cup (\{i_0\} \times S_1)$ with $i = (i_0, i_1)$, and injections in_0, in_1,
- $L = L_0 \uplus L_1$, their disjoint union, with injections j_0, j_1,
- transitions

$$\begin{aligned} t \in Tran \Leftrightarrow & \exists (s, a, s') \in Tran_0.\ t = (in_0(s), j_0(a), in_0(s')) \ \text{ or} \\ & \exists (s, a, s') \in Tran_1.\ t = (in_1(s), j_1(a), in_1(s')). \end{aligned}$$

The construction $T_0 + T_1$ on transition systems T_0, T_1 has injection morphisms $I_0 = (in_0, j_0) : T_0 \to T_0 + T_1$ and $I_1 = (in_1, j_1) : T_1 \to T_0 + T_1$. They together satisfy the universal property required of a *coproduct in a category*; *viz.* . given any morphisms $f_0 : T_0 \to T$ and $f_1 : T_1 \to T$ to a transition system T there is a unique morphism $h : T_0 + T_1 \to T$ such that $h \circ I_0 = f_0$ and $h \circ I_1 = f_1$:

$$\begin{tikzcd} & T & \\ T_0 \arrow[ur, "f_0"] \arrow[r, "I_0"'] & T_0 + T_1 \arrow[u, dashed, "h"] & T_1 \arrow[ul, "f_1"'] \arrow[l, "I_1"] \end{tikzcd}$$

Proposition 6 *Let T_0 and T_1 be transition systems. Then $T_0 + T_1$, with injections $(in_0, j_0), (in_1, j_1)$, is a coproduct in the category of transition systems.*
A state s is reachable in a coproduct iff there is s_0 reachable in T_0 with $s = in_0(s_0)$ or there is s_1 reachable in T_1 with $s = in_1(s_1)$.

The coproduct is not quite of the kind used in modelling the sum of CCS for example, because in the coproduct labels are made disjoint. We look to coproducts in a fibre.

For a labelling set L, each fibre $p^{-1}(L)$ has coproducts. Recall $p^{-1}(L)$ is that subcategory of $\mathbf{T}$ consisting of transition systems over a common labelling set L with morphisms those which project to the identity on L. In form fibre coproducts are very similar to coproducts of transition systems in general—they differ only in the labelling part.

Definition: Let $T_0 = (S_0, i_0, L, Tran_0)$ and $T_1 = (S_1, i_1, L, Tran_1)$ be transition systems over the same labelling set L. Define $T_0 +_L T_1 = (S, i, L, Tran)$ (note it is over the same labelling set) where:
$S = (S_0 \times \{i_1\}) \cup (\{i_0\} \times S_1)$ with $i = (i_0, i_1)$, and injections in_0, in_1, and

$$\begin{aligned} t \in Tran \Leftrightarrow & \exists (s, a, s') \in Tran_0.\ t = (in_0(s), a, in_0(s')) \ \text{ or} \\ & \exists (s, a, s') \in Tran_1.\ t = (in_1(s), a, in_1(s')). \end{aligned}$$

Proposition 7 *Let T_0 and T_1 be transition systems over L. The transition system $T_0 +_L T_1$ with injections $(in_0, 1_L)$ and $(in_1, 1_L)$, as defined above, is a coproduct in the subcategory of transition systems consisting of the fibre $p^{-1}(L)$.*

Neither the coproduct or fibre coproduct of transition systems quite match the kind of sums used in modelling processes, for example, in CCS. The coproduct changes the labels, tagging them so they are disjoint, while the fibre coproduct, seemingly more appropriate because it leaves the labels unchanged, assumes that the transition systems have the same labelling set. A more traditional sum is the following:

Definition: Let T_0 and T_1 be transition systems over L_0 and L_1 respectively. Define their *sum* $T_0 \oplus T_1$ to be $(S, i, L_0 \cup L_1, Tran)$ where
$S = (S_0 \times \{i_1\}) \cup (\{i_0\} \times S_1)$ with $i = (i_0, i_1)$, and injections in_0, in_1, and

$$\begin{aligned} t \in Tran \Leftrightarrow & \exists (s, a, s') \in Tran_0.\ t = (in_0(s), a, in_0(s')) \ \text{ or} \\ & \exists (s, a, s') \in Tran_1.\ t = (in_1(s), a, in_1(s')). \end{aligned}$$

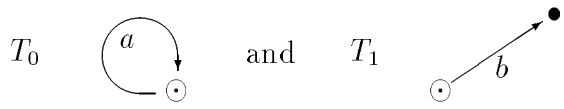
This sum can be understood as a fibre coproduct, but where first we form cocartesian liftings of the inclusion maps into the union of the labelling sets; this simply has the effect of enlarging the labelling sets to a common labelling set, their union, where we can form the fibre coproduct:

Proposition 8 *Let T_0 and T_1 be transition systems over L_0 and L_1 respectively. Let $j_k : L_k \hookrightarrow L_0 \cup L_1$ be the inclusion maps, for $k = 0, 1$. Then*

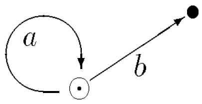
$$T_0 \oplus T_1 \cong T_0\{j_0\} +_{(L_0 \cup L_1)} T_1\{j_1\}.$$

Only coproducts of two transition systems have been considered. All coproducts exist in fibres and in the category of all transition systems. Thus there are indexed sums of transition systems of the kind used in CCS. The sum construction on transition systems is of the form required for CCS when the transition systems are "nonrestarting", *i.e.* have no transitions back to the initial state. In giving and relating semantics we shall be mindful of this fact.

Example: The fibred coproduct $T_0 +_L T_1$ of



both assumed to have the labelling set $L = \{a, b\}$, takes the form:



The sum can behave like T_0 but then on returning to the initial state behave like T_1.

2.2.6 Prefixing

The categorical constructions form a basis for languages of parallel processes with constructs like parallel compositions and nondeterministic sums. The cartesian and cocartesian liftings give rise to restriction and relabelling operations as special cases, but the more general constructions, arising for morphisms in the base category which are truly partial, might also be useful constructions to introduce into a programming language. This raises an omission from our collection of constructions; we have not yet mentioned an operation which introduces new transitions from scratch. Traditionally, in languages like CCS, CSP and Occam this is done with some form of prefixing operation, the effect of which is to produce a new process which behaves like a given process once a specified, initial action has taken place. Given a transition system, the operation of prefixing $a(-)$ introduces a transition, with label a, from a new initial state to the former initial state in a copy of the transition system. One way to define prefixing on transition systems concretely is by:

Definition: Let a be a label (not $*$). Define the *prefix* $aT = (S', i', L', \mathit{Tran}')$ where

$$\begin{aligned}
&S' = \{\{s\} \mid s \in S\} \cup \{\emptyset\},\\
&i' = \emptyset,\\
&L' = L \cup \{a\},\\
&\mathit{Tran}' = \{(\{s\}, b, \{s'\}) \mid (s, b, s') \in \mathit{Tran}\} \cup \{(\emptyset, a, \{i\})\}.
\end{aligned}$$

Because we do not ensure that the prefixing label is distinct from the former labels, prefixing does not extend to a functor on all morphisms of transition systems. However, it extends to a functor on the subcategory of label-preserving morphisms, *i.e.* those morphisms $(\sigma, \lambda) : T \to T'$ between transition systems for which $\lambda : L \hookrightarrow L'$ is an inclusion function. As a special case, prefixing $a(-)$ extends to a functor between fibres $p^{-1}(L) \to p^{-1}(L \cup \{a\})$.

3 A process language

A process language **Proc** and its semantics can be built around the constructions on the category of transition systems. Indeed the process language can be interpreted in all the models we consider. Its syntax is given by

$$t ::= nil \ \mid at \mid t_0 \ \oplus \ t_1 \mid t_0 \times t_1 \mid t \restriction \Lambda \mid t\{\Xi\} \mid x \mid rec\, x.t$$

where a is a label, Λ is a subset of labels and Ξ is a total function from labels to labels. We have seen how to interpret most of these constructions in transition systems, which in particular will yield a labelling set for each term. It is convenient to broaden the understanding of a restriction $t \restriction \Lambda$ so it means the same as $t \restriction \Lambda \cap L$ in the situation where the labelling set L does not include Λ. The denotation of $t\{\Xi\}$ is obtained from the cocartesian lifting with respect to t of the function $\Xi : L \to \Xi L$, so that t with labelling set L is relabelled by Ξ cut down to domain L. The new construction is the recursive construction of the form $rec\, x.t$, involving x a variable over processes. It smooths the presentation that we insist that in a recursive definition $rec\, x.t$ the occurences of x in t are *guarded* in the sense that all occurrences of x in t lie within a prefix term.

The presence of process variables means that the denotation of a term as a transition system is given with respect to an environment ρ mapping process variables to transition systems. We can proceed routinely, by induction on the structure of terms, to give an interpretation of syntactic operations by those operations on transition systems we have introduced, for example we set

> $\mathbf{T}[\![nil]\!]\rho = nil$, for a choice of initial transition systems
> $\mathbf{T}[\![t_0 \oplus t_1]\!]\rho = \mathbf{T}[\![t_0]\!]\rho \oplus \mathbf{T}[\![t_1]\!]\rho$, the nondeterministic sum of section 2.2.5

But how are we to interpret $\mathbf{T}[\![rec\, x.t]\!]\rho$, for an environment ρ, assuming we have an interpretation $\mathbf{T}[\![t]\!]\rho'$ for any environment ρ'?

There are several techniques in use for giving meaning to recursively defined processes and in this section we will discuss two. One approach is to use ω-colimits with respect to some suitable subclass of morphisms in the category of transition systems and use the fact that the operations of the process language can be represented by functors which are continuous in the sense of preserving ω-colimits. For example, all the operations needed to model **Proc** are continuous functors on the subcategory of transition systems with label-preserving monomorphisms—this subcategory has all ω-colimits. However we can work more concretely and choose monomorphisms which are inclusions. In this instance the general method then becomes a mild generalisation of that of fixed points of continuous functions on a complete partial order. The method is based on the observation that transition systems almost form a complete partial order under the relation

$$(S, i, L, tran) \trianglelefteq (S', i', L', tran') \text{ iff } S \subseteq S' \ \& \ i = i \ \& \ L \subseteq L' \ \& \ tran \subseteq tran'$$

associated with the existence of a morphism from one transition system to another based on inclusion of states and labelling sets. Objects of the category of transition systems do not form a set, but they do have least upper bounds of ω-chains

$$T_0 \trianglelefteq T_1 \trianglelefteq \cdots \trianglelefteq T_n \trianglelefteq \cdots$$

of transition systems $T_n = (S_n, i_n, L_n, tran_n)$, for $n \in \omega$; the least upper bound $\bigcup_{n\in\omega} T_n$ is given simply by

$$\bigcup_{n\in\omega} T_n = (\bigcup_{n\in\omega} S_n, i_n, \bigcup_{n\in\omega} L_n, \bigcup_{n\in\omega} tran_n).$$

There is no unique least element, but rather a class of minimal transition systems $(\{i\}, i, \emptyset\ \emptyset)$ for a choice of initial state i. However this is no obstacle to a treatment of guarded recursions based on the order $\trianglelefteq$.

First observe that each operation, prefixing, sum, product, restriction and relabelling has been defined concretely, and in fact each operation is continuous with respect to $\trianglelefteq$. It follows that for an term t, and process variable x, that the operation F, given by

$$F(T) = \mathbf{T}[\![t]\!]\rho[T/x],$$

on transition systems is continuous. Moreover, if x is guarded in t, then for any choice of transition system T, the initial state of $F(T)$ is the same, i say. Consequently, writing $I = (\{i\}, i, \emptyset\ \emptyset)$ we have

$$I \trianglelefteq F(I)$$

and inductively, by monotonicity:

$$I \trianglelefteq F(I) \trianglelefteq F^2(I) \trianglelefteq \cdots \trianglelefteq F^n(I) \trianglelefteq \cdots$$

Write $fix(F) =_{def} \bigcup_{n\in\omega} F^n(I)$. By the continuity of F we see that $fix(F)$ is the $\trianglelefteq$-least fixed point of F. In fact because F is defined from a term in which x is guarded, we can show a uniqueness property of its fixed points:

Definition: For $T = (S, i, L, tran)$ a transition system, define $\mathcal{R}(T)$ to be the transition system $(S', i, L', Tran')$ consisting of states S' reachable from i, with initial state i, and transitions $Tran' = Tran \cap (S' \times L \times S')$ with labelling set L' consisting of those labels appearing in $Tran'$.

Lemma 9 *If T is a transition system for which $T \cong \mathcal{R}(F(T))$, a label-preserving isomorphism, then $T \cong \mathcal{R}(fix(F))$, a label-preserving isomorphism.*

Proof: The proof of this fact depends on several subsidiary definitions and results which we place in Appendix C. $\square$

We can now complete our denotational semantics, the denotation $\mathbf{T}[\![rec\, x.t]\!]\rho$ being taken to be $fix(F)$ where $F(T) = \mathbf{T}[\![t]\!]\rho[T/x]$.

3.1 Operational semantics (version 1)

Alternatively, we can give a structural operational semantics to our language on standard lines. In doing so it is useful to introduce a little notation concerning the combination of labels. For labels a, b define

$$a \times b = \begin{cases} * \text{ if } a = b = *, \\ (a, b) \text{ otherwise} \end{cases}$$

This notation along with the use of idle transitions gives a single compact rule for product. The transitions between states, identified with closed terms, are given by the following rules:

$$\frac{}{at \xrightarrow{a} t} \qquad \frac{}{t \xrightarrow{*} t}$$

$$\frac{t_0 \xrightarrow{a} t_0'}{t_0 \oplus t_1 \xrightarrow{a} t_0'}\ a \neq * \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_0 \oplus t_1 \xrightarrow{a} t_1'}\ a \neq *$$

$$\frac{t_0 \xrightarrow{a} t_0' \quad t_1 \xrightarrow{b} t_1'}{t_0 \times t_1 \xrightarrow{a \times b} t_0 \times t_1'}$$

$$\frac{t \xrightarrow{a} t'}{t \upharpoonright \Lambda \xrightarrow{a} t \upharpoonright \Lambda}\ a \in \Lambda \qquad \frac{t \xrightarrow{a} t'}{t\{\Xi\} \xrightarrow{\Xi(a)} t'\{\Xi\}}$$

$$\frac{t[rec\, x.t/x] \xrightarrow{a} t'}{rec\, x.t \xrightarrow{a} t'}\ a \neq *$$

A closed term t determines a transition system with initial state t consisting of all states and transitions which are reachable from t.

Unfortunately the relationship between the transition systems obtained denotationally and operationally is a little obscure. There are several mismatches. One is that the categorical sum makes states of the two components of a sum disjoint, a property which cannot be shared by the transition system of the operational semantics, essentially because of incidental identifications of syntax. Furthermore, the transition system for recursive processes can lead to transition systems with transitions back to the initial state. As we have seen this causes a further mismatch between the denotational and operational treatment of sums. Indeed the denotational treatment of recursive processes will lead to acyclic transition systems, which are generally not obtained with the present operational semantics. Less problematic is the fact that from the very way it is defined the transition systems obtained operationally must consist only of reachable states and transitions. This property is not preserved by the categorical operation of restriction used in the denotational semantics.

Of course, if we use a coarser relation of equivalence than isomorphism then the two semantics can be related. In the next section, it will be shown that, given any term, there is a strong bisimulation (in the sense of [55]) between the reachable states of the transition system obtained denotationally and those got from the operational semantics.

3.2 Operational semantics (version 2)

The denotational and operational approaches can be reconciled in a simple way. The idea is to modify the operational semantics, to introduce new copies of states where they are required by the denotational semantics. New copies of states are got by tagging terms by 0, 1, or 2. States for the operational semantics are built from closed terms from the syntax extended to include the clauses

$$t ::= \cdots \mid (0, t) \mid (1, t) \mid (2, t).$$

We call such terms *tagged* terms—note they include the ordinary terms.

The modified operational semantics for tagged terms is given by these rules:

$$\frac{t \xrightarrow{a} t'}{(n, t) \xrightarrow{a} (n, t')}$$

$$\frac{}{at \xrightarrow{a} t} \qquad \frac{}{t \xrightarrow{*} t}$$

$$\frac{t_0 \xrightarrow{a} t_0'}{t_0 \oplus t_1 \xrightarrow{a} (0, t_0')}\ a \neq * \qquad \frac{t_1 \xrightarrow{b} t_1'}{t_0 \oplus t_1 \xrightarrow{b} (1, t_1')}\ b \neq *$$

$$\frac{t_0 \xrightarrow{a} t_0' \quad t_1 \xrightarrow{b} t_1'}{t_0 \times t_1 \xrightarrow{a \times b} t_0' \times t_1'}$$

$$\frac{t \xrightarrow{a} t'}{t \upharpoonright \Lambda \xrightarrow{a} t' \upharpoonright \Lambda}\ a \in \Lambda \qquad \frac{t \xrightarrow{a} t'}{t\{\Xi\} \xrightarrow{\Xi(a)} t'\{\Xi\}}$$

$$\frac{t[rec\ x.t/x] \xrightarrow{a} t'}{rec\ x.t \xrightarrow{a} (2, t')}\ a \neq *$$

The first rule expresses that a tagged term has the capabilities of the untagged term. Notice that the former operational semantics is obtained by stripping away the tags, and in fact such a relation is a bisimulation, in the sense of Milner and Park [55], between the transition systems of the two forms of operational semantics.

Now we can establish a close correspondence between the operational and denotational semantics.

Definition: Letting T be the transition system of the operational semantics, with initial state a tagged term t, define

$$\mathcal{O}p(t) = \mathcal{R}(T).$$

Lemma 10 *For any closed tagged term t, the transition system $\mathcal{O}p(t)$ is acyclic.*

Proof: We show this by mapping tagged terms t to $\mid t \mid$ in a strict order $<$ (an irreflexive, transitive relation) in such a way that

$$t \xrightarrow{a} u \ \& \ a \not\equiv * \Rightarrow \mid t \mid < \mid u \mid . \tag{1}$$

It then follows that $\rightarrow^+$ is irreflexive. The full proof, with the definition of $<$, is given in Appendix C. $\square$

Theorem 11 *Let t be a closed term of the process language* **Proc**. *For any environment ρ*

$$\mathcal{O}p(t) \cong \mathcal{R}(\mathbf{T}[\![t]\!]\rho),$$

a label-preserving isomorphism.

Proof: By structural induction we show if t is a term with free variables $x_1, \cdots, x_k$ then for all closed terms $t_1, \cdots, t_k$,

$$\mathcal{O}p(t[t_1/x_1, \cdots, t_k/x_k]) \cong \mathcal{R}(\mathbf{T}[\![t]\!]\rho[\mathcal{O}p(t_1)/x_1, \cdots, \mathcal{O}p(t_k)/x_k]),$$

a label-preserving isomorphism. Henceforth in this proof we will use vector notation, writing *e.g.* $\overline{x}$ for $x_1, \cdots, x_k$ and $\overline{T/x}$ for $T_1/x_1, \cdots, T_k/x_k$.

The basis cases, when t is *nil* or a variable, hold trivially. The case where t is a prefix uses acyclicity of the operational semantics in order to ensure disjointness of the initial state, as does that of sum where, as we have seen, we require components are non-restarting in order for the categorical sum to reflect that given operationally. The denotational and operational semantics of the operations product, restriction and relabelling correspond closely making the proof simple in these cases. The only case of difficulty is that where t has the form $rec\, y.u$:

Assume $rec\, y.u$ has free variables $\overline{x}$ and that $\overline{s}$ are closed terms to instantiate $\overline{x}$. Writing $v = u[\overline{s/x}]$, we observe that from acyclicity (lemma 10) it follows that

$$\mathcal{O}p(rec\, y.v) \cong \mathcal{O}p(v[rec\, y.v/y])$$

—the isomorphism acts so

$$\begin{array}{lcl} rec\, y.v & \mapsto & v[rec\, y.v/y] \\ (2,r) & \mapsto & r \end{array}$$

and is clearly label-preserving. Hence

$$\begin{array}{rcl} \mathcal{O}p(rec\, y.v) & \cong & \mathcal{O}p(v[rec\, y.v/y]) \\ & = & \mathcal{O}p(u[\overline{s/x}, rec\, y.v/y]) \\ & \cong & \mathcal{R}(\mathbf{T}[\![u]\!]\rho[\overline{\mathcal{O}p(s)/x},\ \mathcal{O}p(rec\, y.v)/y]) \end{array}$$

where the latter isomorphism is label-preserving by the induction hypothesis. Thus $\mathcal{O}p(rec\, y.v) \cong \mathcal{R}(F(\mathcal{O}p(rec\, y.v))$, also label-preserving, where

$$F(T) = \mathbf{T}[\![u]\!]\rho[\overline{\mathcal{O}p(s)/x}, T/y].$$

But $\mathbf{T}[\![rec\, y.u]\!]\rho[\overline{\mathcal{O}p(s)/x}] = fix(F)$, and so by lemma 9,

$$\mathcal{O}p(rec\, y.u[\overline{s/x}]) = \mathcal{O}\rho(rec\, y.v) \cong \mathcal{R}(\mathbf{T}[\![rec\, y.u]\!]\rho[\overline{\mathcal{O}\rho(s)/x}]),$$

the label-preserving isomorphism required for this case of the induction. □

There is another way in which the operational and denotational semantics agree. There is a strong bisimulation (in the sense of [55]) between the reachable states of the transition system obtained as the denotational semantics of a term and those got from the operational semantics. In fact, the bisimulation can be expressed as a special kind of morphism on transition systems, called *zig-zag* morphisms by van Bentham—*cf.* the chapter [86].

Definition: A morphism of transition systems $f : T \to T'$ is called a *zig-zag* morphism iff both T and T' have the same labelling set L, the morphism has the form $f = (\sigma, 1_L)$ (*i.e.* f preserves labels) and

$$\sigma(s) \xrightarrow{a} u \text{ in } T' \Rightarrow \exists s'.\ s \xrightarrow{a} s' \text{ in } T'$$

for all states s, u of T and labels a.

For a closed term t, let T_1 be the transition system obtained from the operational semantics (version 1) with initial state t. Write $\mathcal{O}p_1(t)$ for $\mathcal{R}(T_1)$. It is easiest to relate the two forms of operational semantics by the obvious function *untag*, taking a tagged term to its associated term without tags. The isomorphism of theorem 11 then yields a zig-zag morphism relating the denotation of a closed term to its operational semantics (version 1).

Proposition 12 *Let t be a closed term. There is a unique morphism of transition systems*

$$\mathcal{O}p(t) \to \mathcal{O}p_1(t)$$

which takes a state s to $untag(s)$; it is a zig-zag morphism.

Proof: Any derivation according to version 1 of the operational semantics is matched by one according to version 2, and *vice versa.* □

3.3 An example

We will illustrate the different models on an example where the parallel composition is given by the following synchronisation algebra. Labels have the form $a?$, $a!$, a which intuitively can be thought of as representing receiving on channel a ($a?$), sending on channel a ($a!$) and completed synchronisation on channel a (simply a). The synchronisation algebra is given by the following table.

$\bullet$	$*$	$a?$	$a!$	a	$b?$	$b!$	b	$\cdots$
$*$	$*$	$a?$	$a!$	a	$b?$	$b!$	b	
$a?$	$a?$	0	a	0	0	0	0	
$a!$	$a!$	a	0	0	0	0	0	
a	a	0	0	0	0	0	0	
$\vdots$								

The synchronisation algebra is like that for CCS, but instead of labelling successful synchronisations by an anonymous τ they retain some identity. We use $||$ to denote its associated parallel composition.

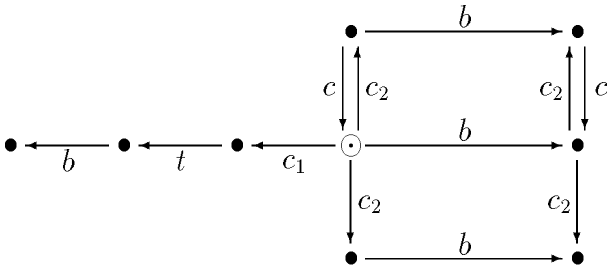
We introduce an example which will reappear in illustrating all the different models. In a form of process algebra it might be described by:

$$SYS = (VM \mid\mid VM' \mid\mid C) \upharpoonright \{b, c, c_1, c_2, t\}$$

where

$$\begin{aligned} VM &= c_2?\, c!\, VM \oplus c_2?\, t!\, VM \\ VM' &= c_1?\, t!\, VM' \oplus b\, nil \\ C &= c_2!\, c?\, C \oplus c_1!\, t?nil. \end{aligned}$$

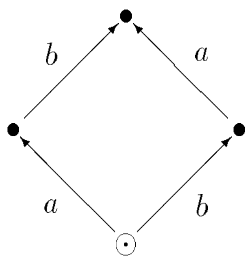
Intuitively, SYS consists of two (rather poorly designed) vending machines VM, VM' in parallel with a customer C. The customer can insert a coin ($c_2!$) to get coffee ($c?$) repeatedly, or insert a coin ($c_1!$) to get tea ($t?$) and stop. The vending machine VM can receive a coin ($c_2?$) to deliver coffee ($c!$) or alternatively receive ($c_2?$) to deliver tea ($t!$)—a customer can't determine which!. The other vending machine VM' is cheaper; it costs less ($c_1?$) to deliver tea ($t!$), but it may breakdown (b). A reasonable model for the system SYS is as the transition system derivable from the operational semantics (version 1) of section 3, illustrated below:



Notice, in particular, the deadlock which can occur if the customer inserts coin c_2 in machine VM, a situation where the customer's request for coffee is met by the machine VM only offering tea. Notice too that the transition system does not capture concurrency in the sense that we expect that a breakdown (b) can occur in parallel with the customer receiving coffee (c) and this is not caught by the transition system. This limitation of transition systems can be seen even more starkly for the two simple terms:

$$a\ b\ nil \oplus b\ a\ nil \qquad\qquad a\ nil\ ||\ b\ nil.$$

both of which can be described by the same transition system, *viz.*



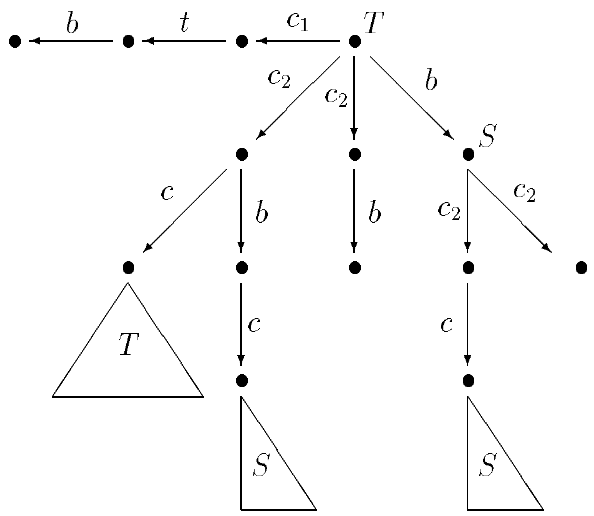
This contrasts the interleaving model of transition systems with noninterleaving models, like Petri nets we shall see later, which represent independence of actions explicitly.

4 Synchronisation trees

We turn to consider another model. It gives rise to our first example illustrating how different models can be related through the help of adjunctions between their associated categories.

In his foundational work on CCS [54], Milner introduced synchronisation trees as a model of parallel processes and explained the meaning of the language of CCS in terms of operations on them. In this section we briefly examine the category of synchronisation trees and its relation to that of labelled transition systems. This illustrates the method by which many other models are related, and the role category theoretic ideas play in formulating and proving facts which relate semantics in one model to semantics in another.

Example: We return to the example of 3.3. As a synchronisation tree SYS could be represented by



where the trees stemming from S and T are repeated as indicated. The synchronisation tree is obtained by unfolding the transition-system of 3.3. Such an unfolding operation arises as an adjoint in the formulation of the models as categories. In moving to synchronisation trees we have lost the cyclic structure of the original transition system, that the computation can repeatedly visit the same state. We can still detect the possibility of deadlock if the customer inserts coin c_2.

As we have seen, a synchronisation tree is a tree together with labels on its arcs. Formally, we define synchronisation trees to be special kinds of labelled transition systems, those for which the transition relation is acyclic and can only branch away from the root.

Definition: A *synchronisation tree* is a transition system (S, i, L, Tran) where

- every state is reachable,
- if $s \xrightarrow{v} s$, for a string v, then v is empty (*i.e.* the transition system is acyclic), and
- $s' \xrightarrow{a} s \ \&\ s'' \xrightarrow{b} s \Rightarrow a = b \ \&\ s' = s''$.

Regarded in this way, we obtain synchronisation trees as a full subcategory of labelled transition systems, with a projection functor to the category of labelling sets with partial functions.

Definition: Write $\mathbf{S}$ for the full subcategory of synchronisation trees in $\mathbf{T}$.

In fact, the inclusion functor $\mathbf{S} \hookrightarrow \mathbf{T}$ has a right adjoint $ts : \mathbf{T} \to \mathbf{S}$ which has the effect of unfolding a labelled transition system to a synchronisation tree.[3]

Definition: Let T be a labelled transition system (S, i, L, Tran). Define $ts(T)$ to be $(S', i', L, \mathit{Tran}')$ where:

- The set S' consists of all finite, possibly empty, sequences of transitions

  $$(t_1, \cdots, t_j, t_{j+1}, \cdots, t_{n-1})$$

  such that $t_j = (s_{j-1}, a_j, s_j)$ and $t_{j+1} = (s_j, a_{j+1}, s_{j+1})$ whenever $1 < j < n$. The element $i' = ()$, the empty sequence.

- The set Tran' consists of all triples (u, a, v) where $u, v \in S'$ and $u = (u_1, \ldots, u_k), v = (u_1, \ldots, u_k,\ (s, a, s'))$, obtained by appending an a transition to u.

Define $\phi : S' \to S$ by taking $\phi(()) = i$ and $\phi((t_1, \cdots, t_n)) = s_n$, where $t_n = (s_{n-1}, a_n, s_n)$.

Theorem 13 *Let T be a labelled transition system, with labelling set L. Then $ts(T)$ is a synchronisation tree, also with labelling set L, and, with the definition above, $(\phi, 1_L) : ts(T) \to T$ is a morphism. Moreover $ts(T), (\phi, 1_L)$ is cofree over T with respect to the inclusion functor $\mathbf{S} \hookrightarrow \mathbf{T}$, i.e. for any morphism $f : V \to T$, with V a synchronisation tree, there is a unique morphism $g : V \to ts(T)$ such that $f = (\phi, 1_L) \circ g$:*

$$\begin{tikzcd} T & ts(T) \arrow[l, "{(\phi,1_L)}"'] \\ & V \arrow[ul, "f"] \arrow[u, "g"'] \end{tikzcd}$$

Proof: Let T be a labelled transition system, with labelling set L. It is easily seen that $ts(T)$ is a labelled transition system with labelling set L and $(\phi, 1_L) : ts(T) \to T$ is a morphism. To show the cofreeness property, let $f = (\sigma, \lambda) : V \to T$ be a morphism from a synchronisation tree V. We require the existence of a unique morphism $g : V \to ts(T)$ such that $f = (\phi, 1_L)g$. The morphism g must

[3] Because we shall be concerned with several categories and functors between them we name the functors in a way that indicates their domain and range.

necessarily have the form $g = (\sigma_1, \lambda)$. The map σ_1 is defined by induction on the distance from the root of states of V, as follows:
On the initial state i_V of V, we take $\sigma_1(i_V) = ()$. For any state v' for which (v, a, v') is a transition of V we take $\sigma_1(v') = \sigma_1(v)$ if $\lambda(a) = *$ and otherwise, in the case where $\lambda(a)$ is defined, take $\sigma_1(v') = \sigma_1(v)((\sigma(v), \lambda(a), \sigma(v'))$.

It follows by induction on the distance of states v from the root that $\sigma(v) = \phi\sigma_1(v)$, and that (σ_1, λ) is the unique morphism such that $f = (\phi, 1_L)g$. (For a very similar, but more detailed, argument see [94].) $\square$

It follows that the operation ts extends to a functor which is right adjoint to the inclusion functor from **S** to **T** and that the morphisms $(\phi, 1_L) : ts(T) \to T$ are the counits of this adjunction (see [50] theorem 2, p.81). This makes **S** a (full) coreflective subcategory of **T**, which implies the intuitively obvious fact that a synchronisation tree T is isomorphic to its unfolding $ts(T)$ (see [50] p.88).

Like transition systems, synchronisation trees have been used to give semantics to languages like CCS and CSP (see *e.g.* [54], [14]). Nondeterministic sums of processes are modelled by the operation of joining synchronisation trees at their roots, a special case of the nondeterministic sum of transition systems. We use $\Sigma_{i \in I} S_i$ for the sum of synchronisation trees indexed by $i \in I$. For the semantics of parallel composition, use is generally made of Milner's "expansion theorem" (see [54]). In our context, the expansion of a parallel composition as a nondeterministic sum appears as a characterisation of the product of synchronisation trees.

Proposition 14 *The product of two synchronisation trees S and T of the form*

$$S \cong \sum_{i \in I} a_i S_i \quad \textit{and} \quad T \cong \sum_{j \in J} b_j T_j.$$

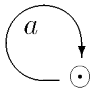
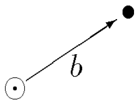
is given by

$$S \times T \cong \sum_{i \in I} (a_i, *) S_i \times T \oplus \sum_{i \in I, j \in J} (a_i, b_j) S_i \times T_j \oplus \sum_{j \in J} (*, b_j) S \times T_j.$$

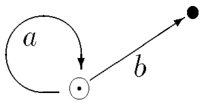
Proof: The fact that the category of synchronisation trees has products and that they are preserved by the unfolding operation ts is a consequence of the general fact that right adjoints preserve limits. In particular, $ts(S \times_T T)$ is a product of the synchronisation trees S and T above; the proof that products of trees have the form claimed follows by considering the sequences of executable transitions of $S \times_T T$. $\square$

The coreflection between transition systems and synchronisation trees is fibrewise in that it restricts to adjunctions between fibres over a common labelling set. For example, for this reason its right adjoint of unfolding automatically preserves restriction (see Appendix B). In fact, via the coreflection **S** inherits a bifibration structure from **T**. As the following example shows, right adjoints, such as the operation of unfolding a transition system to a tree, do not necessarily preserve colimits like nondeterministic sums.

Example: Recall the fibred coproduct $T_0 +_L T_1$ of

T_0  and T_1 

both assumed to have the labelling set $L = \{a, b\}$, is



It is easily seen that $ts(T_0 +_L T_1)$ is not isomorphic to $ts(T_0) +_L ts(T_1)$.

5 Languages

Synchronisation trees abstract away from the looping structure of processes. Now we examine a yet more abstract model, that of languages which further ignore the nondeterministic branching structure of processes.

Definition: A *language* over a labelling set L consists of (H, L) where H is a non-empty subset of strings L^* which is closed under prefixes, *i.e.* if $a_0 \cdots a_{i-1}a_i \in H$ then $a_0 \cdots a_{i-1} \in H$.

Thus for a language (H, L) the empty string ε is always contained in H. Such languages were called *traces* in [33] and for this reason, in the context of modelling concurrency, they are sometimes called *Hoare traces.* They consist however simply of strings and are not to be confused with the traces of Mazurkiewicz, to be seen later.

Example: Refer back to the customer-vending machine example of 3.3. The semantics of SYS as a language (its Hoare traces), loses the nondeterministic structure present in both the transition-system and synchronisation-tree descriptions. The language determined by SYS is

$$\{\varepsilon,\ c_1,\ c_2,\ b,\ c_1t,\ c_2c,\ c_2b,\ bc_2,\ \cdots\}.$$

Lost is the distinction between for instance the two branches of computation c_2b, one which can be resumed by further computation and the other which deadlocks.

Morphisms of languages are partial functions on their alphabets which send strings in one language to strings in another:

Definition: A partial function $\lambda : L \to_* L'$ extends to strings by defining

$$\widehat{\lambda}(sa) = \begin{cases} \widehat{\lambda}(s)\lambda(a) & \text{if } \lambda(a) \text{ defined ,} \\ \widehat{\lambda}(s) & \text{if } \lambda(a) \text{ undefined .} \end{cases}$$

A *morphism* of languages $(H, L) \to (H', L')$ consists of a partial function $\lambda : L \to_* L'$ such that $\forall s \in H.\ \widehat{\lambda}(s) \in H'$.
We write $\mathbf{L}$ for the category of languages with the above understanding of morphisms, where composition is our usual composition of partial functions.

Ordering strings in a language by extension enables us to regard the language as a synchronisation tree. The ensuing notion of morphism coincides with that of languages. This observation yields a functor from $\mathbf{L}$ to $\mathbf{S}$. On the other hand any transition system, and in particular any synchronisation tree, gives rise to a language consisting of strings of labels obtained from the sequences of transitions it can perform. This operation extends to a functor. The two functors form an adjunction from $\mathbf{S}$ to $\mathbf{L}$ (but not from $\mathbf{T}$ to $\mathbf{L}$).

Definition: Let (H, L) be a language. Define $ls(H, L)$ to be the synchronisation tree $(H, \varepsilon, L, tran)$ where

$$(h, a, h') \in \mathit{Tran} \Leftrightarrow h' = ha.$$

Let $T = (S, i, L, \mathit{Tran})$ be a synchronisation tree. Define $sl(T) = (H, L)$ where a string $h \in L^*$ is in the language H iff there is a sequence, possibly empty, of transitions

$$i \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$$

in T such that $h = a_1 a_2 \cdots a_n$. Extend sl to a functor by defining $sl(\sigma, \lambda) = \lambda$ for $(\sigma, \lambda) : T \to T'$ a morphism between synchronisation trees.

Theorem 15 *Let (H, L) be a language. Then $ls(H, L)$ is a synchronisation tree, with labelling set L, and, $1_L : sl \circ ls(H, L) \to (H, L)$ is an isomorphism. Moreover $sl \circ ls(H, L), 1_L$ is cofree over (H, L) with respect to the functor $sl : \mathbf{S} \to \mathbf{L}$, i.e. for any morphism $\lambda : sl(T) \to (H, L)$, with T a synchronisation tree, there is a unique morphism $g : T \to ls(H, L)$ such that $\lambda = 1_L \circ sl(g)$:*

$$\begin{tikzcd} (H, L) & sl \circ ls(H, L) \arrow[l, "1_L"'] \\ & sl(T) \arrow[ul, "\lambda"] \arrow[u, "sl(g)"'] \end{tikzcd}$$

Proof: Each state s of the synchronisation tree T is associated with a unique sequence of transitions

$$i \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$$

with $s_n = s$. Defining $\sigma(s)$ to be $\widehat{\lambda}(a_1 \cdots a_n)$ makes (σ, λ) the unique morphism $g : T \to ls(H, L)$ such that $\lambda = 1_L \circ sl(g)$. $\square$

This demonstrates the adjunction $\mathbf{S} \to \mathbf{L}$ with left adjoint sl and right adjoint ls; the fact that the counit is an isomorphism makes the adjunction a (full) reflection. Let $r : \mathbf{L} \to \mathbf{Set}_*$ be the functor sending a morphism $\lambda : (H, L) \to (H', L')$ of languages to $\lambda : L \to_* L'$. Let $q : \mathbf{S} \to \mathbf{Set}_*$ be the functor sending synchronisation trees to their labelling sets (a restriction of the functor p from transition systems). With respect to these projections the adjunction is fibred.

We can immediately observe some categorical constructions. The fibre product and coproduct are simply intersection and union of languages over the same labelling set. The product of two languages $(H_0, L_0), (H_1, L_1)$ takes the form

$$(\widehat{\pi}_0^{-1} H_0 \cap \widehat{\pi}_1^{-1} H_1, L_0 \times_* L_1),$$

with projections $\pi_0 : L_0 \times_* L_1 \to_* L_0$ and $\pi_1 : L_0 \times_* L_1 \to_* L_1$ obtained from the product in $\mathbf{Set}_*$. The coproduct of languages $(H_0, L_0), (H_1, L_1)$ is

$$(\widehat{j}_0 H_0 \cup \widehat{j}_1 H_1, L_0 \uplus L_1)$$

with injections $j_0 : L_0 \to L_0 \uplus L_1, j_1 : L_1 \to L_0 \uplus L_1$ into the left and right component of the disjoint union. The fibre product and coproduct of languages over the same alphabet are given simply by intersection and union respectively.

The expected constructions of restriction and relabelling arise as (strong) cartesian and cocartesian liftings.

6 Relating semantics

We can summarise the relationship between the different models by recalling the coreflection and reflection (and introducing a little notation to depict such adjunctions):

$$\mathbf{L} \hookrightarrow\!\!\!\!\longrightarrow\!\!\triangleleft\; \mathbf{S} \hookrightarrow\!\!\!\!\longrightarrow\!\!\triangleright\; \mathbf{T}$$

The coreflection and reflection are associated with "inclusions", embedding one model in another—the direction of the embedding being indicated by the hooks on the arrows, whose tips point in the direction of the left adjoint. Each inclusion has an adjoint; the inclusion of the coreflection has a right and that of the reflection a left adjoint. These functors from right to left correspond respectively to losing information about the looping, and then in addition the nondeterministic branching structure of processes.[4]

Such categorical facts are useful in several ways. The coreflection $\mathbf{S} \hookrightarrow \mathbf{T}$ tells us how to construct limits in $\mathbf{S}$ from those it $\mathbf{T}$. In particular, we have seen how the form of products in $\mathbf{S}$ is determined by their simpler form in $\mathbf{T}$. We regard synchronisation trees as transition systems via the inclusion functor, form the limit there and then transport it to $\mathbf{S}$, using the fact that right adjoints preserve limits. Because the adjunctions are fibrewise, the right adjoints also preserve cartesian liftings and left adjoints cocartesian liftings (as is shown in Appendix B, lemma 91). The fact that the embedding functors are full and faithful ensures that they reflect limits and colimits, as well as cartesian and cocartesian morphisms because the adjunctions are fibrewise (lemma 92).

Imagine giving semantics to the process language **Proc** of section 3 in any of the three models. Any particular construct is interpreted as being built up in the same way from universal constructions. For example, product in the process language is interpreted as categorical product, and nondeterministic sum in the language as the same combination of cocartesian liftings and coproduct we use in transition systems. Constructions are interpreted in a uniform manner in any of the different models. Prefixing for languages requires a (straightforward) definition. Recursion requires a separate treatment. Synchronisation trees can be ordered in the same way as transition systems. Languages can be ordered by inclusion. In both cases it is straightforward to give a semantics. With respect to an environment ρ_S from process variables to synchronisation trees we obtain a denotational semantics yielding a synchronisation tree

$$\mathbf{S}[\![t]\!]\rho_S$$

for any process term t. And with respect to an environment ρ_L from process variables to languages the denotational semantics yields a language

$$\mathbf{L}[\![t]\!]\rho_L$$

for a process term t. What is the relationship between the three semantics

$$\mathbf{T}[\![-]\!],\ \ \mathbf{S}[\![-]\!],\ \ \text{and}\ \ \mathbf{L}[\![-]\!]?$$

Consider the relationship between the semantics in transition systems and synchonisation trees. Letting ρ be an environment from process variables, to transition systems, the two semantics are related by

$$ts(\mathbf{T}[\![t]\!]\rho) = \mathbf{S}[\![t]\!]ts \circ \rho$$

[4]**Warning:** We use the term "coreflection" to mean an adjunction in which the unit is a natural isomorphism, or equivalently (by theorem 1, p.89 of [50]) when the left adjoint is full and faithful. Similarly, "reflection" is used here to mean an adjunction for which the counit is a natural isomorphism, or equivalently when the right adjoint is full and faithful. While the same uses can be found in the literature, they are not entirely standard.

for any process term t. This is proved by structural induction on t. The cases where t is a product or restriction follow directly from preservation properties of right adjoints. The other cases require special, if easy, argument. For example, the fact that

$$ts(\mathbf{T}[\![t_0 \oplus t_1]\!]\rho) = \mathbf{S}[\![t_0 \oplus t_1]\!] ts \circ \rho$$

depends on $\mathbf{T}[\![t_0]\!]\rho, \mathbf{T}[\![t_1]\!]\rho$ being nonrestarting, a consequence of acyclicity (lemma 10) shown earlier. The case of recursion requires the $\trianglelefteq$-continuity of the unfolding functor ts. A similar relationship,

$$sl(\mathbf{S}[\![t]\!]\rho) = \mathbf{L}[\![t]\!] sl \circ \rho$$

for a process term t, and environment ρ to synchronisation trees, holds between the two semantics in synchronisation trees and languages. This time the structural induction is most straightforward in the cases of nil, nondeterministic sum and relabelling (because of the preservation properties of the left adjoint sl). However, simple arguments suffice for the other cases.

In summary, the operations on processes are interpreted in a uniform manner (with the same universal constructions) in the three different semantics. The preservation properties of adjoints are useful in relating the semantics. Less directly, a knowledge of what we can and cannot expect to be preserved automatically provides useful guidelines in itself. The failure of a general preservation property can warn that the semantics of a construct can only be preserved in special circumstances. For instance, we cannot expect a right adjoint like ts to always preserve a colimit, like a nondeterministic sum. Accordingly, the semantics of sums is only preserved by ts by virtue of a special circumstance, that the transition systems denoted are nonrestarting. The advantages of a categorical approach become more striking when we turn to the more intricate models of the non-interleaving approach to concurrency, but where again the same universal constructions will be used.

7 Trace languages

All the models we have considered so far have identified concurrency, or parallelism, with nondeterministic interleaving of atomic actions. We turn now to consider models where concurrency is modelled explicitly in the form of independence between actions. In some models, like Mazurkiewicz traces, the relation of independence is a basic notion while in others, like Petri nets, it is derived from something more primitive. The idea is that if two actions are enabled and also independent then they can occur concurrently, or in parallel. Models of this kind are sometimes said to capture "true concurrency", a convenient though regrettably biased expression. They are also often called "noninterleaving models" though this again is inappropriate; as we shall see, Petri nets can be described as forms of transition systems. A much better term is "independence models" for concurrent computation, though this is not established. Because in such models the independence of actions is not generally derivable from an underlying property of their labels, depending rather on which occurrences are considered, we will see an important distinction basic to these richer models. They each have a concept of *events* distinguished from that of *labels*. Events are to be thought of as atomic actions which can support a relation of independence. Events can then bear the further structure of having a label, for instance signifying which channel or which process they belong to.

A greater part of the development of these models is indifferent to the extra labelling structure we might like to impose, though of course restriction and relabelling will depend on labels. Our treatment of the models and their relationship will be done primarily for the unlabelled structures. Later we will adjoin labelling and provide semantics in terms of the various models and discuss their relationship.

7.1 A category of trace languages

The simplest model of computation with an in-built notion of independence is that of Mazurkiewicz trace languages. They are languages in which the alphabet also possesses a relation of independence. As we shall see, this small addition has a striking effect in terms of the richness of the associated structures. It is noteworthy that, in applications of trace languages, there have been different understandings of the alphabet; in Mazurkiewicz 's original work the alphabet is thought of as consisting of events (especially events of a Petri net), while some authors have instead interpreted its elements as labels, for example standing for port names. This remark will be elaborated later on in section 7.2.

Definition: A *Mazurkiewicz trace language* consists of (M, L, I) where L is a set, $I \subseteq L \times L$ is a symmetric, irreflexive relation called the *independence* relation, and M is a nonempty subset of strings L^* such that

- *prefix closed*: $sa \in M \Rightarrow s \in M$ for all $s \in L^*, a \in L$,
- *I-closed*: $sabt \in M \;\&\; aIb \Rightarrow sbat \in M$ for all $s, t \in L^*, a, b \in L$,
- *coherent*: $sa \in M \;\&\; sb \in M \;\&\; aIb \Rightarrow sab \in M$ for all $s \in L^*, a, b \in L$.

The alphabet L of a trace language (M, L, I) can be thought of as the set of actions of a process and the set of strings as the sequences of actions the process can perform. Some actions are independent of others. The axiom of I-closedness expresses a consequence of independence: if two actions are independent and can occur one after the other then they can occur in the opposite order. The axiom of coherence is not generally imposed. We find it convenient (though not essential for a great deal that follows), and besides, like I-closedness, it seems to follow from an intuitive understanding of what independence means; it says if two actions are independent and both can occur from the same state then they can occur one after the other, in either order. Given that some actions are independent of others, it is to be expected that some strings represent essentially the same computation as others. For example, if a and b are independent then both strings ab and ba represent the computation of a and b occurring concurrently. More generally, two strings $sabt$ and $sbat$ represent the same computation when a and b are independent. This extends to an equivalence relation between strings, the equivalence classes of which are called *Mazurkiewicz traces.* There is an associated preorder between strings of a trace language which induces a partial order on traces.

Definition: Let (M, L, I) be a trace language. For $s, t \in M$ define $\approx$ to be the smallest equivalence relation such that

$$sabt \approx sbat \text{ if } aIb$$

for $sabt, sbat \in M$. Call an equivalence class $\{s\}_{\approx}$, for $s \in M$, a *trace.* For $s, t \in M$ define

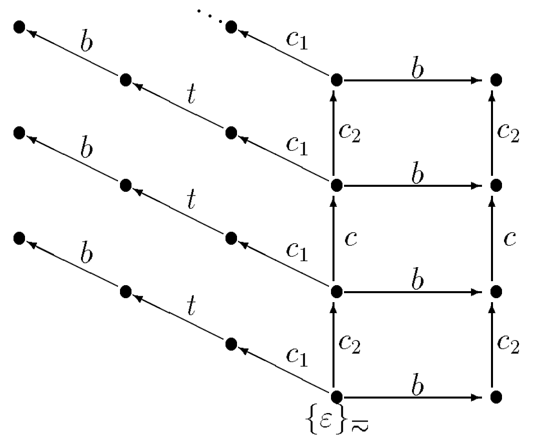
$$s \lesssim t \Leftrightarrow \exists u.\ su \approx t.$$

Proposition 16 *Let (M, L, I) be a trace language with trace equivalence $\approx$. If $su \in M$ and $s \approx s'$ then $s'u \in M$ and $su \approx s'u$. The relation $\lesssim$ of the trace language is a preorder. Its quotient $\lesssim / \approx$ by the equivalence relation $\approx$ is a partial order on traces.*

Example: The independence relation of Mazurkiewicz allows us to express the concurrency we remarked on earlier in example 3.3. By asserting that the independence relation I is the smallest such that

$$b\ I\ c \quad \text{and} \quad b\ I\ c_2,$$

corresponding to the idea that a breakdown b can occur in parallel with the customer receiving coffee, the language of example 5 collapses under the identifications of trace equivalence to give the following ordering $\lesssim / \approx$ on traces:



We have drawn the traces as points and drawn an arrow $\xrightarrow{a}$ from a trace $\{s\}_{\approx}$ to a trace $\{sa\}_{\approx}$. Although the potential concurrency of b and c, and b and c_2 is caught by the independence relation, this trace-language semantics, like the language semantics before, is blind to the fact that the system can deadlock after the customer inserts a coin c_2. To make such a distinction we would have to distinguish the two kinds of occurrences of c_2, regarding them as different events.

The partial order $\lesssim / \approx$ of a trace language can be associated with a partial order of causal dependencies between event occurrences. This structure will be investigated in the next section.

Morphisms between trace languages are morphisms between the underlying languages which preserve independence:

Definition: A *morphism* of trace languages $(M, L, I) \to (M', L', I')$ consists of a partial function $\lambda : L \to_* L'$ which

- *preserves independence*: aIb & $\lambda(a)$ defined & $\lambda(b)$ defined $\Rightarrow \lambda(a)I'\lambda(b)$ for all $a, b \in L$,
- *preserves strings*: $s \in M \Rightarrow \widehat{\lambda}(s) \in M'$ for all strings s.

It is easy to see that morphisms of trace languages preserve traces and the ordering between them:

Proposition 17 *Let $\lambda : (M, L, I) \to (M', L', I')$ be a morphism of trace languages. If $s \lesssim t$ in the trace language (M, L, I) then $\widehat{\lambda}(s) \lesssim \widehat{\lambda}(t)$ in the trace language (M', L', I').*

Definition: Write **TL** for the category of trace languages with composition that of partial functions.

7.2 Constructions on trace languages

We examine some categorical constructions on trace languages. The constructions generalise from those on languages but with the added consideration of defining independence.

Let (M_0, L_0, I_0) and (M_1, L_1, I_1) be trace languages. Their *product* is (M, L, I) where $L = L_0 \times_* L_1$, the product in $\mathbf{Set}_*$, with projections $\pi_0 : L \rightarrow_* L_0$ and $\pi_1 : L \rightarrow_* L_1$, with

$$\begin{aligned} aIb \Leftrightarrow & (\pi_0(a), \pi_0(b) \text{ defined } \Rightarrow \pi_0(a) I_0 \pi_0(b)) \ \& \\ & (\pi_1(a), \pi_1(b) \text{ defined } \Rightarrow \pi_1(a) I_1 \pi_1(b)), \end{aligned}$$

and

$$M = \widehat{\pi}_0^{-1} M_0 \cap \widehat{\pi}_1^{-1} M_1.$$

Their *coproduct* is (M, L, I) where $L = L_0 \uplus L_1$, the disjoint union, with injections $j_0 : L_0 \to L, j_1 : L_0 \to L$, the relation I satisfies

$$\begin{aligned} aIb \Leftrightarrow & \exists a_0, b_0.\ a_0 I_0 b_0 \ \&\ a = j_0(a_0) \ \&\ b = j_0(b_0) \text{ or} \\ & \exists a_1, b_1.\ a_1 I_1 b_1 \ \&\ a = j_1(a_1) \ \&\ b = j_1(b_1) \end{aligned}$$

and

$$M = \widehat{j}_0 M_0 \cup \widehat{j}_1 M_1.$$

What about restriction and relabelling? Restriction appears again as a cartesian lifting of an inclusion between labelling sets. Its effect is simply to cut-down the language and independence to the restricting set. However, the relabelling of a trace language cannot always be associated with a cocartesian lifting. To see this consider a function $\lambda : \{a, b\} \to \{c\}$ sending both a, b to c. If a trace language T has $\{a, b\}$ as an alphabet and has a, b independent then, λ cannot be a morphism of trace languages, and hence no cocartesian lifting of λ with respect to the trace language T; because independence is irreflexive, independence cannot be preserved by λ.[5] The difficulty stems from our implicitly regarding the alphabet of a Mazurkiewicz trace language as a set of labels of the kind used in the operations of restriction and relabelling. Although the alphabet can be taken to have this nature, it was not the original intention of Mazurkiewicz. Here it is appropriate to discuss the two ways in which trace languages are used to the model parallel processes.

One way is to use trace languages in the same manner as languages. This was implicitly assumed in our attempts to define the relabelling of a trace language, and in the example above. Then a process, for example in CCS, denotes a trace language, with alphabet the labels of the process. This regards symbols of the alphabet of a trace language as labels in a process algebra. As we have

[5] If however we instead project to the category of *sets with independence* $\mathbf{Set}_I$ we obtain a fibration and cofibration—see section 8.3.3, in particular proposition 40.

seen in the treatment of interleaving models labels can be understood rather generally; they are simply tags to distinguish some actions from others. However this general understanding of the alphabet conflicts with this first approach. As the independence relation is then one between labels, once it is decided that say a and b are independent in the denotation of a process then they are so throughout its execution. However, it is easy to imagine a process where at some stage a and b occur independently and yet not at some other stage. To remedy this some have suggested that the independence relation be made to depend on the trace of labels which has occurred previously. But even with this modification, the irreflexivity of the independence relation means there cannot be independent occurrences with the same label; in modelling a CCS process all its internal τ events would be dependent!

The other approach is to regard the alphabet as consisting, not of labels of the general kind we have met in process algebras, but instead as consisting of *events*. It is the events which possess an independence relation and any distinctions that one wishes to make between them are then caught through an extra labelling function from events to labels. True, this extra level of labelling complicates the model, but the distinction between events and the labels they can carry appears to be fundamental. It is present in the other models which capture concurrency directly as independence. This second view fits with that of Mazurkiewicz's trace-language semantics of Petri nets. When we come to adjoin the extra structure of labels, restriction will again be associated with a cartesian lifting and relabelling will reappear as a cocartesian lifting.

There remains the question of understanding the order $\lesssim / \approx$ of trace languages. We shall do this through a representation theorem which will show that $\lesssim / \approx$ can be understood as the subset ordering between configurations of an event structure.

8 Event structures

There is most often no point in analysing the precise places and times of events in a distributed computation. What is generally important 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. 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. One thing missing from such descriptions is the phenomenon of nondeterminism. To model nondeterminism we adjoin further structure in the form of a conflict relation between events to express how the occurrence of certain events rules out the occurrence of others. Here we shall assume that events exclude each other in a binary fashion, though variants of this have been treated.

Definition: Define an *event structure* to be a structure $(E, \leq, \#)$ 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

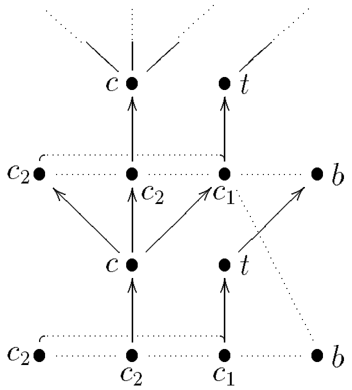
$$\{e' \mid e' \leq e\} \text{ is finite,}$$
$$e \# e' \leq e'' \Rightarrow e \# e''$$

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

Say two events $e, e' \in E$ are *concurrent*, and write $e \; co \; e'$, iff $\neg(e \leq e'$ or $e' \leq e$ or $e \# e')$. Write $\mathrel{W}$ for $\# \cup 1_E$, *i.e.* the reflexive closure of the conflict relation.

The finiteness assumption restricts attention to discrete processes where an event occurrence depends only on finitely many previous occurrences. The axiom on the conflict relation expresses that if two events causally depend on events in conflict then they too are in conflict. Note that events of an event structure correspond to event *occurrences*.

Example: As an illustration, we examine how to represent the process SYS of 3.3 as an event structure. Strictly speaking SYS is represented as the *labelled* event structure, drawn below, where the event ocurrences (the events of the event structure) appear as "•", labelled to indicate their nature. The sequential nature of the components VM, VM' and C imposes a partial order of causal dependency $\leq$, the immediate steps of which are drawn as upwards arrows, and nondeterminism shows up as conflict, generated by the relation indicated by dotted lines.



In fact this labelled event structure is that obtained from the denotational semantics following the general scheme of section 11.2.

Guided by our interpretation we can formulate a notion of computation state of an 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—and also that

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

—no two conflicting events can occur together in the same computation.

Definition: Let $(E, \leq, \#)$ be an event structure. Define its *configurations*, $\mathcal{D}(E, \leq, \#)$, to consist of those subsets $x \subseteq E$ which are

- *conflict-free:* $\forall e, e' \in x.\ \neg(e \# e')$ and
- *downwards-closed:* $\forall e, e'.\ e' \leq e \in x \Rightarrow e' \in x$.

In particular, define $\lceil e \rceil = \{e' \in E \mid e' \leq e\}$. (Note that $\lceil e \rceil$ is a configuration as it is conflict-free.)

Write $\mathcal{D}^0(E, \leq, \#)$ for the set of finite configurations.

The important relations asociated with an event structure can be recovered from its finite configurations (or indeed similarly from its configurations):

Proposition 18 *Let $(E, \leq, \#)$ be an event structure. Then*

- $e \leq e' \Leftrightarrow \forall x \in \mathcal{D}^0(E, \leq, \#).\ e' \in x \Rightarrow e \in x.$
- $e \# e' \Leftrightarrow \forall x \in \mathcal{D}^0(E, \leq, \#).\ e \in x \Rightarrow e' \notin x.$
- $e \ co \ e' \Leftrightarrow \exists x, x' \in \mathcal{D}^0(E, \leq, \#).\ e \in x \ \& \ e' \notin x \ \& \ e' \in x' \ \& \ e \notin x' \ \& \ x \cup x' \in \mathcal{D}^0(E, \#, \leq).$

Events manifest themselves as atomic jumps from one configuration to another, and later it will follow that we can regard such jumps as transitions in an asynchronous transition system.

Definition: Let $(E, \leq, \#)$ be an event structure. Let x, x' be configurations. Write

$$x \xrightarrow{e} x' \Leftrightarrow e \notin x \ \& \ x' = x \cup \{e\}.$$

Proposition 19 *Two events e_0, e_1 of an event structure are in the concurrency relation co iff there exist configurations x, x_0, x_1, x' such that:*

$$\begin{tikzcd}
& x' & \\
x_0 \arrow[ur, "e_1"] & & x_1 \arrow[ul, "e_0"'] \\
& x \arrow[ul, "e_0"] \arrow[ur, "e_1"'] &
\end{tikzcd}$$

8.1 A category of event structures

We define morphisms on event structures as follows:

Definition: Let $ES = (E, \leq, \#)$ and $ES' = (E', \leq', \#')$ be event structures. A *morphism* from ES to ES' consists of a partial function $\eta : E \to_* E'$ on events which satisfies

$$x \in \mathcal{D}(ES) \Rightarrow \eta x \in \mathcal{D}(ES') \ \& \\ \forall e_0, e_1 \in x.\ \eta(e_0), \eta(e_1) \text{ both defined } \ \& \ \eta(e_0) = \eta(e_1) \Rightarrow e_0 = e_1.$$

A morphism $\eta : ES \to ES'$ between event structures expresses how behaviour in ES determines behaviour in ES'. The partial function η expresses how the occurrence of an event in ES implies the simultaneous occurrence of an event in ES'; the fact that $\eta(e) = e'$ can be understood as expressing that the event e' is a "component" of the event e and, in this sense, that the occurrence of e implies the simultaneous occurrence of e'. If two distinct events in ES have the same image in ES' under η then they cannot belong to the same configuration.

Morphisms of event structures preserve the concurrency relation. This is a simple consequence of proposition 19, showing how the concurrency relation holding between events appears as a "little square" of configurations.

Proposition 20 *Let ES be an event structure with concurrency relation co and ES' an event structure with concurrency relation co'. Let $\eta : ES \to ES'$ be a morphism of event structures. Then, for any events e_0, e_1 of ES,*

$$e_0 \ co\ e_1 \ \& \ \eta(e_0), \eta(e_1) \ \textit{both defined} \Rightarrow \eta(e_0) \ co' \ \eta(e_1).$$

Morphisms between event structures can be described more directly in terms of the causality and conflict relations of the event structure:

Proposition 21 *A morphism of event structures from $(E, \leq, \#)$ to $(E', \leq', \#')$ is a partial function $\eta : E \to_* E'$ such that*

(i) $\eta(e)$ defined $\Rightarrow \lceil \eta(e) \rceil \subseteq \eta \lceil e \rceil$ and

(ii) $\eta(e_0), \eta(e_1)$ both defined $\ \& \ \eta(e_0) \mathbb{W}' \eta(e_1) \Rightarrow e_0 \mathbb{W} e_1$.

The category of event structures possesses products and coproducts useful in modelling parallel compositions and nondeterministic sums.

Proposition 22 *Let $(E_0, \leq_0, \#_0)$ and $(E_1, \leq_1, \#_1)$ be event structures. Their coproduct in the category* **E** *is the event structure $(E_0 \uplus E_1, \leq, \#)$ where*

$$e \leq e' \Leftrightarrow (\exists e_0, e_0'.\ e_0 \leq_0 e_0' \ \& \ j_0(e_0) = e \ \& \ j_0(e_0') = e') \ or \\ (\exists e_1, e_1'.\ e_1 \leq_1 e_1' \ \& \ j_1(e_1) = e \ \& \ j_1(e_1') = e')$$

and

$$\# = \#_0 \cup \#_1 \cup (j_0 E_0) \times (j_1 E_1),$$

with injections $j_0 : E_0 \to E_0 \uplus E_1, j_1 : E_1 \to E_0 \uplus E_1$ *the injections of* E_0 *and* E_1 *into their disjoint union.*

It is tricky to give a direct construction of product on event structures. However, a construction of the product of event structures will follow from the product of trace languages and the coreflection from event structures to trace languages (see corollary 39), and we postpone the construction till then.

8.2 Domains of configurations

Viewing computation states as such subsets, progress in a computation is measured by the occurrence of more events. Let $x, y \in \mathcal{D}(E)$ for an event structure E. If $x \subseteq y$ then x can be regarded as a subbehaviour of y. The relation of inclusion between configurations is an information order of the sort familiar from denotational semantics, but special in that more information corresponds to more events having occurred. It is easy to see that the order $(\mathcal{D}(E), \subseteq)$ has least upper bounds, when they exist, given as unions, and that the order is a cpo with least element the empty set. The domains associated with event structures turn out to be familiar. (Proofs of the following characterisations can be found in [98].)

The simplest characterisation of the domains represented by prime event structures starts by observing that an event e in an event structure corresponds to the configuration $\lceil e \rceil$. Such elements are characterised as being *complete primes* and domains of configurations have the property that every element is the least upper bound of these special elements.

Definition: Let $(D, \sqsubseteq)$ be a partial order with least upper bounds of subsets X written as $\bigsqcup X$ when they exist.

Say D is *bounded complete* iff all subsets $X \subseteq D$ which have an upper bound in D have a least upper bound $\bigsqcup X$ in D.

Say D is *coherent* iff all subsets $X \subseteq D$ which are pairwise bounded (*i.e.* such that all pairs of elements $d_0, d_1 \in X$ have upper bounds in D) have least upper bounds $\bigsqcup X$ in D. (Note that coherence implies bounded completeness.)

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

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

for any set X for which $\bigsqcup X$ exists.

D is *prime algebraic* iff

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

for all $x \in L$. If furthermore the sets

$$\{p \sqsubseteq q \mid p \textit{ is a complete prime}\}$$

are always finite when q is a complete prime, then D is said to be *finitary*.

If D is bounded complete and prime algebraic it is a *prime algebraic domain.*

Theorem 23 *Let E be a event structure. The partial order $(\mathcal{D}(E), \subseteq)$ is a coherent, finitary, prime algebraic domain; the complete primes are the set $\{\lceil e \rceil \mid e \in E\}$.*

Proof: See [64]. □

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

Theorem 24 *Let $(D, \sqsubseteq)$ be a coherent, finitary, prime algebraic domain. Define $(P, \leq, \#)$ to consist of P, the complete primes of D, ordered by*

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

and with relation

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

for $p, p' \in P$. Then $(P, \leq, \#)$ is an event structure, with $\phi : (D, \sqsubseteq) \cong (\mathcal{D}(P, \leq, \#), \subseteq)$ giving an isomorphism of partial orders where $\phi(d) = \{p \sqsubseteq d \mid p \text{ is a complete prime}\}$ with inverse $\theta : \mathcal{D}(P, \leq, \#) \to (D, \sqsubseteq)$ given by $\theta(x) = \bigsqcup x$.

Proof: See [64]. □

Event structures and coherent, finitary prime algebraic domains are equivalent; one can be used to represent the other. Such domains are familiar in another guise. (Recall that the dI-domains of Berry are distributive algebraic cpos in which every finite element only dominates finitely many elements [8].)

Theorem 25 *The finitary, prime algebraic domains are precisely the dI-domains of Berry.*

Proof: See [98] or [93]. □

Following Girard, call a function *linear* iff it is *stable* in the sense of Berry (*i.e.* preserves bounded meets) and preserves joins when they exist. The *covering* relation between configurations of an event structures is given by

$$x \to x' \Leftrightarrow_{def} \exists e.\ x \xrightarrow{e} x',$$

for configurations x, x'.

Theorem 26 *The category of event structures* $\mathbf{E}$ *is equivalent to the subcategory of coherent dI-domains with morphisms those linear functions* $f : D \to D'$ *which further satisfy*

$$x \to x' \Rightarrow f(x) = f(x') \text{ or } f(x) \to f(x').$$

Proof: A proof can be found in the report [92]. $\square$

8.3 Event structures and trace languages

8.3.1 A representation theorem

Throughout this section assume (M, L, I) is a trace language. In this section we study the preorder

$$s \lesssim t \Leftrightarrow \exists u.\ su \approx t$$

of a trace language and show that its quotient $\lesssim / \approx$ can be represented by the finite configurations of an event structure.

We use $a, b, c, \ldots$ for symbols in L and $s, t, u, \ldots$ for strings in L^*. Write $N(b, s)$ for the number of occurrences of b in the string s. We write $a \in s$ to mean a occurs in s, *i.e.* $N(a, s) > 0$. As an abbreviation, we write sIt if aIb for every symbol a in s and b in t.

Events of (M, L, I), to be thought of as event occurrences, are taken to be equivalence classes of nonempty strings with respect to the equivalence relation $\sim$ now defined.

Definition: The relation $\sim$ is the smallest equivalence relation on nonempty strings such that

$$\begin{array}{rcll} sa & \sim & sba & \text{if } bIa, \text{ and} \\ sa & \sim & ta & \text{if } s \approx t \end{array}$$

for $sa, sba, ta \in M$.

The next lemma yields an important technique for reasoning about trace languages.

Lemma 27 *Suppose* $sa, ta \in M$.

$$\neg(aIb)\ \&\ sa \sim ta \Rightarrow N(b, s) = N(b, t).$$

Proof: Assume $\neg(aIb)$. It is sufficient to verify the lemma's claim in the case of $sa \sim ta$ where either

(i) ($t = sc$ and cIa) or

(ii) $s \approx t$.

If (i) then $b \neq c$ (because one is independent of a and one not) so $N(b,t) = N(b,sc) = N(b,s)$. If (ii) then $N(b,s) = N(b,t)$ because the number of occurrences of a symbol is invariant under $\approx$. $\square$

As $\neg(aIa)$ the lemma in particular yields

$$sa \sim ta \Rightarrow N(a,s) = N(a,t)$$

for $sa, ta \in M$. Thus different occurrences of the same symbol in a string of M are associated with different events:

Proposition 28 *Suppose s_0a, s_1a are prefixes of $t \in M$ such that $s_0a \sim s_1a$. Then $s_0a = s_1a$.*

We can now show how the preorder of trace languages coincides with the order of inclusion on the associated sets of events:

Definition: Let $s \in M$. Define the events of s, to be

$$ev(s) = \{\{u\}_\sim \mid u \text{ is a nonempty prefix of } s\}.$$

Lemma 29 *Let $s, t \in M$. Then*

$$s \lesssim t \Leftrightarrow ev(s) \subseteq ev(t).$$

Proof: "$\Rightarrow$": We show the claim that, letting $s, t \in M$,

$$s \approx t \Rightarrow ev(s) = ev(t),$$

from which "$\Rightarrow$" follows. It is sufficient to establish this claim for the case where $s = uabv$ and $t = ubav$ with aIb. However, then

$$\begin{aligned} ev(s) &= ev(u) \cup \{\{ua\}_\sim, \{uab\}_\sim\} \cup \{\{uabv'\}_\sim \mid v' \text{ is a nonempty prefix of } v\} \\ &= ev(u) \cup \{\{uba\}_\sim, \{ub\}_\sim\} \cup \{\{ubav'\}_\sim \mid v' \text{ is a nonempty prefix of } v\} \\ &= ev(t). \end{aligned}$$

"$\Leftarrow$": This is proved by induction on s. The basis $s = \varepsilon$ is obvious. Assume $ev(s) \subseteq ev(t)$ where $s = s'a$ and, inductively, that $s' \lesssim t$, *i.e.* $s'u' \approx t$ for some u'. Because $\{s'a\}_\sim \in ev(s)$ certainly $\{s'a\}_\sim \in ev(t) = ev(s'u')$. It follows that $u' = u_0au_1$ for some u_0, u_1 such that $s'a \sim s'u_0a$. (We cannot have $\{s'a\}_\sim \in ev(s')$ by proposition 28 above.) We must have u_0Ia as otherwise there would be $b \in u_0$ with $\neg(bIa)$ and $N(b,s') < N(b,s'u_0)$ contradicting lemma 27. Hence $t \approx s'u_0au_1 \approx s'au_0u_1$ making $s'a \lesssim t$. This proves the induction step. $\square$

The next lemma shows that incompatibility between traces stems from a lack of independence between events.

Lemma 30 *Let $s, t \in M$.*

$$\exists u \in M.\ ev(u) = ev(s) \cup ev(t)$$

iff

$$\forall v \in M, a, b \in L.\ \{va\}_\sim \in ev(s)\ \&\ \{vb\}_\sim \in ev(t) \Rightarrow a(I \cup 1_L)b.$$

Proof: "if": This implication is proved by induction on t. The basis case when $t = \varepsilon$ is obvious. To prove the induction step assume $t = t'a \in M$ and that

$$\{va\}_\sim \in ev(s)\ \&\ \{vb\}_\sim \in ev(t) \Rightarrow a = b \text{ or } aIb$$

for all $v \in M$ and $a, b \in L$. Inductively we assume that there is $u' \in M$ for which $ev(u') = ev(s) \cup ev(t')$. If $\{t'a\}_\sim \in ev(s)$ then $ev(u') = ev(s) \cup ev(t'a)$ as required. Assume otherwise that $\{t'a\}_\sim \notin ev(s)$. By lemma 29, $t' \precsim u'$ so $t'w \approx u'$ for some string w. Assume w has the form $b_1 \dots b_k$. By lemma 29 and proposition 28, we necessarily have $\{t'b_1 \dots b_i\}_\sim \in ev(s)$ for all i where $0 < i \leq k$. Let

$$u_i = t'b_1 \dots b_i a$$

for $0 \leq i \leq k$. We show by induction on i that $u_i \in M$ and $u_i \sim t'a$. Certainly this holds for the basis when $u_0 = t'a$. To establish the induction step assume $i > 0$ and, inductively, that $u_{i-1} = t'b_1 \dots b_{i-1}a \in M$. Because $\{t'b_1 \dots b_{i-1}a\}_\sim \in ev(t)$ and $\{t'b_1 \dots b_{i-1}b_i\}_\sim \in ev(s)$ by assumption we have $a = b_i$ or aIb_i. However $a \neq b_i$ because otherwise $u_i = t'b_1 \dots b_{i-1}b_i$ making $\{t'a\}_\sim \in ev(s)$, contrary to our assumption. Now that we know aIb_i the coherence axiom on trace languages ensures

$$u_i = t'b_1 \dots b_{i-1}b_i a \in M.$$

In addition

$$u_i = t'b_1 \dots b_{i-1}b_i a \sim t'b_1 \dots b_{i-1}a = u_{i-1} \sim t'a.$$

Thus by induction we have established that

$$u_i \in M \text{ and } u_i \sim t'a$$

for $0 \leq i \leq k$. In particular

$$u_k = t'b_1 \dots b_k a = u'a \in M \text{ and } u_k \sim t'a.$$

It follows that

$$\begin{aligned} ev(u_k) &= ev(u') \cup \{\{t'a\}_\sim\} \\ &= ev(s) \cup ev(t') \cup \{\{t'a\}_\sim\} \\ &= ev(s) \cup ev(t'a). \end{aligned}$$

We can thus maintain the induction hypothesis whether or not $\{t'a\}_\sim \in ev(s)$. This establishes the "if" direction of the lemma by induction.

"only if": Assume that $ev(u) = ev(s) \cup ev(t)$ for $s, t, u \in M$ and that the "only" if direction fails to hold. I.e. suppose $\{va\}_\sim \in ev(s)$, $\{vb\}_\sim \in ev(t)$ $a \neq b$ and $\neg(aIb)$ for $v \in M$ and $a, b \in L$. Then either $u = u_0 a u_1 b u_2$ with $va \sim u_0 a$ and $vb \sim u_0 a u_1 b$, or the symmetric case with a and b interchanged. In the former case we observe that

$$\begin{array}{rcll} N(a,v) & = & N(a,u_0) & \text{as } va \sim u_0 a \text{ by lemma 27,} \\ & < & N(a, u_0 a u_1). & \end{array}$$

But $N(a,v) = N(a, u_0 a u_1)$ as $vb \sim u_0 a u_1 b$ by lemma 27. This, and the similar contradictions obtained in the symmetric case, demonstrate the absurdity of our supposition, and thus the "only if" direction of the lemma. $\square$

Remark The above lemma implies that the preorder $\lesssim$ satisfies a finite form of coherence in the sense that any pairwise bounded finite subset has a least upper bound. The coherence axiom on trace languages was essential in proving the "if" direction of the equivalence. Without the coherence axiom, a finite form of bounded completeness can be demonstrated, *i.e.* a finite set with an upper bound has a least upper bound. More precisely it can be shown without use of the coherence axiom that

$$s \lesssim u \;\&\; t \lesssim u \Rightarrow \exists v, w.\; u \approx vw \;\&\; ev(v) = ev(s) \cup ev(t)$$

for all $s, t, u \in M$, from which the finite form of bounded completeness follows.

The following lemma says that each event has a $\lesssim$-minimum representative.

Lemma 31 *For all events e there is $sa \in e$ such that*

$$\forall ta \in e.\; sa \lesssim ta.$$

Proof: We use a characterisation of $\sim$ in the proof. Define

$$sa \prec_1 ta \text{ iff } (t = sb \;\&\; bIa) \text{ or } s \approx t$$

for sa, ta in M. Take $\sim_1 =_{def} \prec_1 \cup \prec_1^{-1}$. Then it is easily seen that $\sim = (\sim_1)^*$.

Let e be an event. Choose sa a $\lesssim$-minimal element of e. We show by induction on k that

$$sa(\sim_1)^k ta \Rightarrow sa \lesssim ta \tag{1}$$

for $ta \in e$. As $\sim = (\sim_1)^*$ the lemma follows.

The basic case, where $k = 0$, holds trivially. Assume inductively that (1) holds for k. If $sa(\sim_1)^{k+1}ta$ then $sa(\sim_1)^k ua \sim_1 ta$ for some $ua \in M$. From the induction hypothesis we obtain

$$sa \lesssim ua.$$

If $ua \prec_1 ta$ then $(t = ub \;\&\; bIa)$ for some b or $u \approx t$, and in either case $ua \lesssim ta$ giving $sa \lesssim ta$, as required to maintain the induction hypothesis. The rub comes if $ua(\prec_1)^{-1}ta$ and this relation holds through $u = tb$ and bIa for some b. Gathering facts, we see

$$sa \sim tba \text{ and } sa \lesssim tba \text{ with } bIa$$

and that we require $sa \lesssim ta$.

By lemma 29 we get

$$ev(sa) \subseteq ev(tba) = ev(tab) = ev(ta) \cup \{\{tab\}_\sim\}.$$

Thus if $\{tab\}_\sim \notin ev(s)$ we obtain $ev(sa) \subseteq ev(ta)$ and hence the required $sa \lesssim ta$, by lemma 29.

We will show by contradiction that $\{tab\}_\sim \notin ev(s)$. Suppose otherwise, that $\{tab\}_\sim \in ev(s)$. Then

$$s = s_0 b s_1 \text{ where } s_0 b \sim tab.$$

Suppose $c \in s_1$ and $\neg(cIb)$. Then

$$N(c, t) > N(c, s_0)$$

which is impossible. Consequently $s_1 Ib$. Thus $sa = s_0 b s_1 a \sim s_0 s_1 ba \sim s_0 s_1 a$. The fact that $s_0 s_1 ab \approx sa$ contradicts the $\lesssim$-minimality of sa. From this contradiction we deduce $\{tab\}_\sim \notin ev(s)$ from which, as remarked, the required $sa \lesssim ta$ follows. $\square$

The minimum representatives are used in defining the event structure associated with a trace language.

Definition: Let $T = (M, L, I)$ be a trace language.
Define

$$tle(M, L, I) = (E, \leq, \#)$$

where

- E is the set of events of (M, L, I),
- $\leq$ is a relation between events e, e' given by $e \leq e'$ iff $e \in ev(s)$ where sa is a minimum representative of e', and

- the relation $e \# e'$ holds between events iff

$$\exists e_0, e_0'.\ e_0 \#_0 e_0' \ \ \&\ e_0 \leq e\ \&\ e_0' \leq e'$$

where, by definition,

$$e_0 \#_0 e_0' \text{ iff } \exists v, a, b.\ va \in e_0\ \&\ vb \in e_0'\ \&\ \neg(a(I \cup 1_L)b).$$

Furthermore, define $\lambda_T : E \to L$ by taking $\lambda_T(\{sa\}_\sim) = a$. (From the definition of $\sim$, it follows that λ_T is well-defined as a function.)

Proposition 32 *Let $T = (M, L, I)$ be a trace language. Then the structure $tle(T) = (E, \leq, \#)$ given by definition 8.3.1 is an event structure for which*

$$\begin{array}{lll} e \leq e' & \text{iff} & \forall s \in M.\ e' \in ev(s) \Rightarrow e \in ev(s) \\ e \# e' & \text{iff} & \forall s \in M.\ e \in ev(s) \Rightarrow e' \notin ev(s). \end{array}$$

Proof: The required facts follow by considering minimum representatives of events. □

We now present the representation theorem for trace languages. We write $(M/\approx, \lesssim/\approx)$ for the partial order obtained by quotienting the preorder $\lesssim$ by its equivalence $\approx$.

Theorem 33 *Let $T = (M, L, I)$ be a trace language. Let $tle(T) = (E, \leq, \#)$. There is an order isomorphism*

$$Ev : (M/\approx,\ \lesssim/\approx) \to \mathcal{D}^0(E, \leq, \#)$$

where $Ev(\{s\}_\approx) = ev(s)$.

Moreover, for $s \in M$, $x \in \mathcal{D}^0(E, \leq, \#)$ and $a \in L$,

$$(\exists e.\ ev(s) \xrightarrow{e} x \text{ in } \mathcal{D}^0(E, \leq, \#)\ \&\ \lambda_T(e) = a) \Leftrightarrow (sa \in M\ \&\ ev(sa) = x).\quad (\dagger)$$

Proof: Let $s \in M$. By the "only if" direction of lemma 30 it follows that $ev(s)$ is a conflict-free subset of events. By lemma 31, $ev(s)$ is downwards-closed with respect to $\leq$. The fact that Ev is well-defined, 1-1, order preserving and reflecting follows from lemma 29. To establish that Ev is an isomorphism it suffices to check Ev is onto. To this end we first prove ($\dagger$).

The "$\Leftarrow$" direction of the equivalence ($\dagger$) follows directly, as follows. Assume $sa \in M$, and $ev(sa) = x$. Then taking $e = \{sa\}_\approx$ yields an event for which $ev(s) \xrightarrow{e} x$ and $\lambda_T(e) = a$. To show "$\Rightarrow$", assume $ev(s) \xrightarrow{e} x$ and $\lambda_T(e) = a$. Let ta be a minimum representative of the event e. As x is downwards-closed

$$ev(t) \subseteq ev(s).$$

Because x is conflict-free we meet the conditions of lemma 30 ("if" direction, with s for s and ta for t) and obtain the existence of $u \in M$ such that

$$ev(u) = ev(s) \cup ev(ta) = x.$$

Hence $s \lesssim u$, *i.e.* $sw \approx u$ for some string w. But $ev(s) \xrightarrow{e} ev(sw)$, so w must be a with $sa \in e$.

Now a simple induction on the size of $x \in \mathcal{D}^0(E, \leq, \#)$ shows that there exists $s \in M$ for which $ev(s) = x$. From this it follows that Ev is onto, and consequently that Ev is an order isomorphism. $\square$

The representation theorem for trace languages establishes a connection between trace languages and the *pomset* languages of Pratt [76]. Via the representation theorem, each trace of a trace language $T = (M, L, I)$ corresponds to a labelled partial order of events (a *partially ordered multiset* or *pomset*)—the partial order on events in the trace is induced by that of the event structure and the labelling function is λ_T. The trace language itself then corresponds to a special kind of pomset language; it is special chiefly because the concurrency relations in the pomsets arise from a single independence relation on the alphabet of labels, so consequently pomsets of traces have no *autoconcurrency*—no two concurrent events have the same label. (See [9], [80] and [30] for more details.)

Via the representation theorem we can see how to read the concurrency relation of an event structure in trace-language terms:

Proposition 34 *For a trace language $T = (M, L, I)$ the construction $tle(M, L, I)$ is an event structure in which the concurrency relation satisfies*

$$e \ co \ e' \text{ iff } \exists va, vb \in M.\ va \in e \ \& \ vb \in e' \ \& \ aIb. \qquad (\dagger)$$

Proof: We show $(\dagger)$.
"if": Assume $va \in e, vb \in e'$ with aIb. Then certainly va, vb are compatible as traces making $\neg e \# e'$. Moreover $e, e' \notin ev(v)$ (by *e.g.* proposition 28) ensuring neither $e \leq e'$ nor $e' \leq e$, whence $e \ co \ e'$.
"only if": Assuming $e \ co \ e'$ there are distinct configurations $x = (\lceil e \rceil \setminus \{e\}) \cup (\lceil e' \rceil \setminus \{e'\})$ and $x_1 = x \cup \{e\}, x_2 = x \cup \{e'\}, y = x \cup \{e, e'\}$. From the representation theorem 33 there is $v \in M$ such that $ev(v) = x$. Assume $\lambda_T(e) = a$ and $\lambda_T(e') = b$. By $(\dagger)$ of the representation theorem, as $x \xrightarrow{e} x_1$ we obtain $va \in M$ with $ev(va) = x_1$. It follows that $va \in e$. Again by $(\dagger)$ of the representation theorem, this time because $x_1 \xrightarrow{e'} y$ we obtain $vab \in M$ with $ev(vab) = y$. It follows that $vab \in e'$. Similarly, it can be shown that $vb \in M$ and $vb \in e'$. Because both vab and vb are representatives of the event e', it follows directly that $vb \sim vab$. If $\neg(aIb)$ then lemma 27 would imply $N(a, v) = N(a, va)$. But this is clearly absurd, yielding aIb. We have produced the $va, vb \in M$ required. $\square$

8.3.2 A coreflection

The representation theorem extends to a coreflection between the categories of event structures and trace languages.

Definition: Let ES be an event structure with events E. Define $etl(ES)$ to be (M, E, co), where $s = e_1 \dots e_n \in M$ iff there is a chain

$$\emptyset \xrightarrow{e_1} x_1 \xrightarrow{e_2} x_2 \dots \xrightarrow{e_n} x_n$$

of configurations of ES.
Let η be a morphism of event structures $\eta : ES \to ES'$. Define $etl(\eta) = \eta$.

Proposition 35 *etl is a functor* $\mathbf{E} \to \mathbf{TL}$.

Proof: The only nontrivial part of the proof is that showing that η is a morphism from $etl(E)$ to $etl(E')$ provided η is a morphism $ES \to ES'$. However, this follows from the proposition 20 and the observation that if a sequence of events s is associated with a chain of configurations in ES then $\hat{\eta}(s)$ is associated with a chain of configurations in ES'. $\square$

The function λ_T, for T a trace language, will be the counit of the adjunction.

Proposition 36 *Let $T = (M, L, I)$ be a trace language. Then,*

$$\lambda_T : etl \circ tle(T) \to T$$

is a morphism of trace languages.

Proof: Let $e_1 e_2 \cdots e_n$ be a string in the trace language $etl \circ tle(T)$. Then there is a chain of configurations of the event structure $tle(T)$

$$\emptyset \xrightarrow{e_1} \{e_1\} \xrightarrow{e_2} \{e_1, e_2\} \cdots \xrightarrow{e_n} \{e_1, e_2, \cdots, e_n\}.$$

By repeated use of (†) in the representation theorem 33, we obtain that $\widehat{\lambda_T}(e_1 e_2 \cdots e_n) \in M$. If $e \; co \; e'$, for events e, e', then by proposition 34, it follows directly that $\lambda_T(e) I \lambda_T(e')$. Thus $\lambda_T : etl \circ tle(T) \to T$ is a morphism of trace languages. $\square$

Lemma 37 *Let $ES = (E, \leq, \#)$ be an event structure, such that $etl(ES) = (M, E, co)$. Let $\lambda : etl(ES) \to T'$ be a morphism in* $\mathbf{TL}$*. If $\lambda(e)$ is defined then for all $se, s'e \in M$*

$$\hat{\lambda}(se) \sim \hat{\lambda}(s'e)$$

in $T' = (M', L', I')$.

Proof: It suffices to consider the following two cases.
The first case is where we assume $s = ue_0e_1v$ and $s' = ue_1e_0v$ where $u, v \in E^*$, $e_0, e_1 \in E$, e_0 *co* e_1 in ES: In this case e_0 and e_1 are independent in $etl(ES)$. But then $\lambda(e_0)I'\lambda(e_1)$ in T' if both defined (from properties of morphisms in **TL**), and hence $\hat{\lambda}(ue_0e_1v) \sim \hat{\lambda}(ue_1e_0v)$ in T'.
The second case arises when $s = s'e'$ for some $e' \in E$ such that e *co* e' in ES: In this case e and e' are independent in $etl(ES)$. But then $\lambda(e)I'\lambda(e')$ in T' if $\lambda(e')$ is defined and hence $\hat{\lambda}(se) \sim \hat{\lambda}(s'e)$. □

Theorem 38 *Let $T' = (M', L', I')$ be a trace language. Then the pair $etl \circ tle(T')$, $\lambda_{T'}$, is cofree over T' with respect to the functor etl. That is, for any event structure ES and morphism $\lambda : etl(ES) \to T'$ there is a unique morphism $\eta : ES \to tle(T')$ such that $\lambda = \lambda_{T'} \circ etl(\eta)$.*

Proof: Let $ES = (E, \leq, \#)$, $tle(T') = (E', \leq', \#')$ and $etl(ES) = (M, E, co)$. Define $\eta : E \to^* E'$ by

$$\eta(e) = \begin{cases} * & \text{if } \lambda(e) = * \\ \{\hat{\lambda}(se)\}_\sim, & \text{where } se \in M, \text{ if } \lambda(e) \neq * \end{cases}$$

It follows from lemma 37 that η is a well-defined partial function from E to E'. We need to prove that

(a) η is a morphism $ES \to tle(T')$

(b) $\lambda = \lambda_{T'} \circ \eta$

(c) η is unique satisfying (a) and (b).

(a): To prove (a), that η is a morphism, it suffices by proposition 21 to prove (i) and (ii) below.
(i) For every $e \in E$, if $\eta(e)$ is defined then $\lceil \eta(e) \rceil \subseteq \eta(\lceil e \rceil)$
Choose $se \in M$ such that the occurrences in s equal $\lceil e \rceil$ (in ES). Assume $x'a' \in M'$ such that

$$\{x'a'\}_\sim \leq \{\hat{\lambda}(se)\}_\sim \text{ in } tle(T'). \tag{$*$}$$

We have to prove the existence of $e_0 \in \lceil e \rceil$ in E such that $\{x'a'\}_\sim = \eta(e_0)$. But from $(*)$ we may choose a minimal prefix s_0e_0 of se such that $x'a' \sim \hat{\lambda}(s_0e_0)$, with $e_0 \in \lceil e \rceil$ from which we conclude the desired property.
(ii) For all $e, e' \in E$. $\eta(e) \mathbb{W}' \eta(e') \Rightarrow e \mathbb{W} e'$
Suppose $\neg e \mathbb{W} e'$ and $\eta(e), \eta(e')$ are both defined. There are essentially two cases to consider, one where e *co* e' and the other where $e < e'$ (or symmetrically $e' < e$). Firstly assume e *co* e' in ES. Then

$$eIe' \text{ in } etl(ES) \;\&\; se \in M \;\&\; se' \in M,$$

for some $s \in M$. Applying the morphism λ, we obtain

$$\lambda(e)I'\lambda(e') \text{ in } T' \;\&\; \hat{\lambda}(se) \in M' \;\&\; \hat{\lambda}(se') \in M.$$

But now

$$\eta(e)\ co\ \eta(e') \text{ in } tle(T')$$

from proposition 34.
Secondly, assuming $e < e'$ in ES, there are $s, s' \in E^*$ such that

$$ses'e' \in M.$$

Applying λ,

$$\hat{\lambda}(ses'e') \in M'.$$

Thus

$$\eta(e) \in ev(\hat{\lambda}(ses'e'))\ \&\ \eta(e') \in ev(\hat{\lambda}(ses'e')),$$

from which it follows that $\neg\eta(e)\#'\eta(e')$ in $tle(T')$, by proposition 32. Furthermore, from $ses'e' \in M$, we get $\eta(e) \neq \eta(e')$; the assumption $\eta(e) = \eta(e')$ implies $\lambda(e) = \lambda(e')$, but $\hat{\lambda}(se) \sim \hat{\lambda}(ses'e')$ contradicts lemma 27. This completes the proof of (a).
(b): If $\lambda(e) = *$ then $\eta(e) = *$, so $(\lambda_{T'} \circ \eta)(e) = *$. If $\lambda(e)$ defined then $\eta(e) = \{\hat{\lambda}(se)\}_\sim$ for some $se \in M$. This implies $\lambda_{T'}(\eta(e)) = \lambda(e)$ by the definition of $\lambda_{T'}$. Hence $\lambda = \lambda_{T'} \circ \eta$.
(c): We now show the uniqueness of η. Assume η' is any morphism from E to $tle(T)$, such that $\lambda_{T'} \circ \eta' = \lambda$. We want to show $\eta(e) = \eta'(e)$ for all $e \in E$. Let $x \xrightarrow{e} x \cup \{e\}$ in ES, and assume inductively that η and η' agree on all elements of x. Firstly, from the assumption $\lambda_{T'} \circ \eta' = \lambda$ we get $\eta'(e)$ defined iff $\lambda(e)$ defined (since $\lambda_{T'}$ is total) and hence iff $\eta(e)$ defined. So, assume $\eta'(e)$ is defined and equal to e'. Then $\eta'(x) \xrightarrow{e'} \eta'(x \cup \{e\})$ in $tle(T')$ (since η' morphism) and $\lambda_{T'}(e') = \lambda(e)$. However, from the representation theorem for trace languages, it follows that there is exactly one event in $tle(T')$ satisfying these requirements—the one picked by η, and hence $\eta(e) = \eta'(e)$. $\square$

Corollary 39 *The operation tle on trace languages extends to a functor, right adjoint to etl, forming a coreflection* $\mathbf{E} \hookrightarrow \mathbf{TL}$*; the functor tle sends the morphism* $\lambda : T \to T'$ *to* $\eta : tle(T) \to tle(T')$ *acting on events* $\{sa\}_\sim$ *of* $tle(T)$ *so that*

$$\eta(\{sa\}_\sim) = \begin{cases} * & \text{if } \lambda(a) \text{ undefined },\\ \{\hat{\lambda}(sa)\}_\sim & \text{if } \lambda(a) \text{ defined }. \end{cases}$$

Proof: It follows from theorem 38 that *tle* extends to a functor, acting as described, so that the pair of functors form an adjunction. From the proof of theorem 38, the unit of this adjunction at ES is the morphism

$$\eta : ES \to tle \circ etl(ES)$$

given by $\eta(e) = \{se\}_\sim$, where se is a possible sequence of events of ES. It is easy to see that η is an isomorphism with inverse $\eta^{-1} : tle \circ etl(ES) \to ES$ such that

$$\eta^{-1}(\{se\}_\sim) = e.$$

$\square$

The coreflection expresses the sense in which the model of event structures "embeds" in the model of trace languages. Because of the coreflection we can restrict trace languages to those which are isomorphic to images of event structures under *etl* and obtain a full subcategory of trace languages equivalent to that of event structures.

The existence of a coreflection from event structures to trace languages has the important consequence of yielding an explicit product construction on event structures, which is not so easy to define directly. The product of event structures E_0 and E_1 can be obtained as

$$tle(etl(E_0) \times etl(E_1)),$$

that is by first regarding the event structures as trace languages, forming their product as trace languages, and then finally regarding the result as an event structure again. That this result is indeed a product of E_0 and E_1 follows because the right adjoint *tle* preserves limits and the unit of the adjunction is a natural isomorphism (*i.e.* from the coreflection). In a similar way we will be able to obtain the product of event structures from that of nets, asynchronous transition systems, or indeed more general event structures, from the coreflections between categories of event structures and these models.

8.3.3 A reflection

The existence of a *coreflection* from the category of event structures to the category of trace languages might seem surprising, at least when seen alongside the analogous interleaving models, where we might think of trace languages as the analogue of languages. There is a *reflection* from the category of languages to the category of synchronisation trees.

This mismatch can be reconciled by recalling the two ways of regarding trace languages (*cf.* the discussion of section 7.2). One is that where the alphabet of a trace language is thought of a consisting of events; this view is adopted in establishing the coreflection. Alternatively, the alphabet can be thought of as a set of labels. With the latter view a more correct analogy is:

- Labelled event structures generalise synchronisation trees.
- Trace languages generalise languages.

As we will see shortly, this analogy can be formalised in a diagram of adjunctions.

In order to define the appropriate category of labelled event structures we first define the category $\mathbf{Set}_I$ of sets with independence. It consists of objects (L, I) where L is a set and I, the independence relation, is a binary, irreflexive relation on L, and morphisms $(L, I) \to (L', I')$ to be partial functions $\lambda : L \to_* L'$ which preserve independence in the sense that

$$aIb \;\&\; \lambda(a) \text{ defined } \;\&\; \lambda(b) \text{ defined } \Rightarrow \lambda(a) I' \lambda(b);$$

composition is that of partial functions.

The category of labelled event structures $\mathcal{L}_I(\mathbf{E})$ has objects $(ES, l : (E, co) \to (L, I))$ where ES is an event structure with events E, and concurrency relation co, and the labelling function $l : (E, co) \to (L, I)$ is a total function in $\mathbf{Set}_I$ (which therefore sends concurrent events to independent labels). We remark that one way an ordinary set L can be regarded as a set with independence is as $(L, L \times L \setminus 1_L)$. The restriction on labelling functions to such sets with independence amounts to the commonly used restriction of banning *autoconcurrency* [25]. A morphism in $\mathcal{L}_I(\mathbf{E})$ has the form

$$(\eta, \lambda) : (ES, l : (E, co) \to (L, I)) \to (ES', l' : (E', co') \to (L', I'))$$

and consists of a morphism of event structures

$$\eta : ES \to ES'$$

and a morphism

$$\lambda : (L, I) \to (L', I')$$

in $\mathbf{Set}_I$ such that

$$l' \circ \eta = \lambda \circ l.$$

The right adjoint of a reflection is $\mathcal{E} : \mathbf{TL} \to \mathcal{L}_I(\mathbf{E})$ defined as follows:

Definition: Let $T = (M, L, I)$ be a trace language. Define $\mathcal{E}(M, L, I)$ to be $(E, \leq, \#, \lambda_T)$ where $(E, \leq, \#)$ is the event structure $tle(T)$ and $\lambda_T : (E, co) \to (L, I)$ in $\mathbf{Set}_I$ is given by the counit at T of the coreflection between (unlabelled) event structures and trace languages.

Let $\lambda : T \to T'$ be a morphism of trace languages. Define $\mathcal{E}(\lambda)$ to be $(tle(\lambda), \lambda)$.

In constructing a left adjoint we make use of a relabelling operation on trace languages, where the relabelling function preserves independence. The operation is described in the following proposition.[6]

Proposition 40 *Let $\lambda : (L', I') \to (L, I)$ be a morphism in $\mathbf{Set}_I$. Let (M', L', I') be a trace language. Define $\lambda_!(M', L', I')$ to be (M, L, I) where M is the smallest prefix-closed, I'-closed and coherent subset (as in the definition of trace languages) containing $\hat{\lambda} M'$. Then $\lambda : (M', L', I') \to \lambda_!(M', L', I')$ is a morphism of trace languages.*

[6] In fact, proposition 40 shows how to construct cocartesian liftings of the functor projecting morphisms $\lambda : (M', L', I') \to (M, L, I)$ in $\mathbf{TL}$ to morphisms $\lambda : (L', I') \to (L, I)$ in $\mathbf{Set}_I$.

Let (ES, l) be a labelled event structure in $\mathcal{L}_I(\mathbf{E})$. Under the coreflection the event structure ES can be a regarded as a trace language $etl(ES)$ over an alphabet consisting of its events. Proposition 40 provides a morphism

$$etl(ES) \to l_! \circ etl(ES),$$

used in defining the left-adjoint of the reflection $\mathcal{T} : \mathcal{L}_I(\mathbf{E}) \to \mathbf{TL}$, given by:

Definition: Let $(ES, l) \in \mathcal{L}_I(\mathbf{E})$, where $ES = (E, \leq, \#)$ and $l : (E, co) \to (L, I)$ in $\mathbf{Set}_I$. Define

$$\mathcal{T}(ES, l) = l_! \circ etl(ES)$$

For $(\eta, \lambda) : (ES, l) \to (ES', l')$ a morphism of $\mathcal{L}_I(\mathbf{E})$, define $\mathcal{T}(\eta, \lambda) = \lambda$.

The proof that $\mathcal{E}$ and $\mathcal{T}$ constitute a reflection uses the coreflection of section 8.3.2. It hinges on the pun in which a set of events is regarded simultaneously as a set of labels.

Theorem 41 *$\mathcal{E} : \mathbf{TL} \to \mathcal{L}_I(\mathbf{E})$ and $\mathcal{T} : \mathcal{L}_I(\mathbf{E}) \to \mathbf{TL}$ are functions with $\mathcal{T}$ left adjoint to $\mathcal{E}$. In fact, if $T = (M, L, I)$ is a trace language then*

$$1_L : \mathcal{T}\mathcal{E}(T) \to T$$

is the counit at T making the adjunction a reflection.

Proof: It is easy to check $\mathcal{E}, \mathcal{T}$ are functors. The fact that $\mathcal{T}\mathcal{E}(T) = T$ follows from the representation theorem (theorem 33), and ensures that $1_L : \mathcal{T}\mathcal{E}(T) \to T$ is a morphism in $\mathbf{TL}$, for any $T \in \mathbf{TL}$ with labelling set L.

Let $(ES, l) \in \mathcal{L}_I(\mathbf{E})$, with $l : (E, co) \to (L', I)$, and $T = (M, L, I) \in \mathbf{TL}$. We show the cofreeness property, that for any $\lambda : \mathcal{T}(ES, l) \to T$ there is a unique morphism $(\eta, \lambda) : (ES, l) \to \mathcal{E}(T)$ such that

$$\begin{tikzcd}
T & \mathcal{T}\mathcal{E}(T) \arrow[l, "1_L"'] \\
& \mathcal{T}(ES, l) \arrow[ul, "\lambda"] \arrow[u, "\mathcal{T}(\eta,\lambda)=\lambda"']
\end{tikzcd}$$

commutes. This follows from a corresponding cofreeness property associated with the coreflection $\mathbf{E} \hookrightarrow \mathbf{TL}$, as we now show.

First note, the cocartesian morphism

$$l : etl(ES) \to l_! \circ etl(ES) = \mathcal{T}(ES)$$

composes with

$$\lambda : \mathcal{T}(ES) \to T$$

to yield a morphism

$$\lambda \circ l : etl(ES) \to T.$$

By definition, $(\eta, \lambda) : (ES, l) \to \mathcal{E}(T) = (tle(T), \lambda_T)$ is a morphism in $\mathcal{L}_I(\mathbf{E})$ iff $\eta : ES \to tle(T)$ is a morphism in $\mathbf{E}$ and $\lambda_T \circ \eta = \lambda \circ l$. This is equivalent to

$\eta : ES \to tle(T)$ is a morphism in $\mathbf{E}$ such that the following diagram commutes:

$$\begin{tikzcd} T & etl \circ tle(T) \arrow[l, "\lambda_T"'] \\ & etl(ES) \arrow[ul, "\lambda \circ l"] \arrow[u, "\eta = etl(\eta)"'] \end{tikzcd}$$

But the coreflection between $\mathbf{E} \hookrightarrow \mathbf{TL}$ ensures the existence of a unique such $\eta : ES \to tle(T)$. $\square$

The reflection

$$\mathbf{L} \hookrightarrow \mathbf{S}$$

between the interleaving models of Hoare languages and synchronisation trees, is paralleled by the reflection

$$\mathbf{TL} \hookrightarrow \mathcal{L}_I(\mathbf{E})$$

between the noninterleaving models of labelled event structures and Mazurkiewicz trace languages. The strings in the Hoare languages are generalised to Mazurkiewicz traces. We can view the relationship in another way which relates to Pratt's work. A Mazurkiewicz trace corresponds to a pomset which has no autoconcurrency (no two concurrent events share the same label). Mazurkiewicz trace languages correspond to particular kinds of pomset languages, ones which are, in particular, associated with a global independence relation between actions. (See [9] for a precise characterisation.)

In more detail we have these reflections and coreflections:

$$\begin{tikzcd} \mathbf{L} \arrow[r, hook] \arrow[d, hook] & \mathbf{S} \arrow[d, hook] \\ \mathbf{TL} \arrow[r, hook] & \mathcal{L}_I(\mathbf{E}) \end{tikzcd}$$

The vertical coreflections have not been explained. Their left adjoints identify a synchronisation tree with a labelled event structure (the events are arcs), and a language with a trace language. In both cases the independence relation is taken to be empty.

What model is to generalise both transition systems and labelled event structures? A suitable model would consist of labels attached to certain structures; a fitting structure should allow loops in the behaviour and have events on which it is possible to interpret a relation of independence. There are several candidates for the appropriate structures, and we turn now to consider one of the earliest used.

9 Petri nets

Petri nets are a well-known model of parallel computation. They come in several variants, and we choose one which fits well with the other models of computation we have described (a good all-round reference on Petri nets is [2]). Roughly, a Petri net can be thought of as a transition system where, instead of a transition occurring from a single global state, an occurrence of an event is imagined to affect only the conditions in its neighbourhood. The independence of events becomes a derived notion; two events are independent if their neighbourhoods of conditions do not intersect. As the definition of a *Petri net* (or simply *net*) we take:

Definition: A *Petri net* consists of $(B, M_0, E, pre, post)$ where

> B is a set of *conditions*, with *initial marking* M_0 a nonempty subset of B,
> E is a set of *events*, and
> $pre : E \to \mathcal{P}ow(B)$ is the *precondition* map such that $pre(e)$ is nonempty for all $e \in E$,
> $post : E \to \mathcal{P}ow(B)$ is the *postcondition* map such that $post(e)$ is nonempty for all $e \in E$.

A Petri net comes with an initial marking consisting of a subset of conditions which are imagined to hold initially. Generally, a *marking*, a subset of conditions, formalizes a notion of global state by specifying those conditions which hold. Markings can change as events occur, precisely how being expressed by the transitions

$$M \xrightarrow{e} M'$$

events e determine between markings M, M'. In defining this notion it is convenient to extend events by an "idling event".

Definition: Let $N = (B, M_0, E, pre, post)$ be a Petri net with events E.
Define $E_* = E \cup \{*\}$.
We extend the pre and post condition maps to $*$ by taking

$$pre(*) = \emptyset, \quad post(*) = \emptyset.$$

Notation: Whenever it does not cause confusion we write ${}^\bullet e$ for the preconditions $pre(e)$ and $e^\bullet$ for the postconditions, $post(e)$, of $e \in E_*$. We write ${}^\bullet e^\bullet$ for ${}^\bullet e \cup e^\bullet$.

Definition: Let $N = (B, M_0, E, pre, post)$ be a net.
For $M, M' \subseteq B$ and $e \in E_*$, define

$$M \xrightarrow{e} M' \text{ iff } {}^\bullet e \subseteq M \;\&\; e^\bullet \subseteq M' \;\&\; M \setminus {}^\bullet e = M' \setminus e^\bullet.$$

Say $e_0, e_1 \in E_*$ are *independent* iff ${}^\bullet e_0^\bullet \cap {}^\bullet e_1^\bullet = \emptyset$.

A marking M of N is said to be *reachable* when there is a sequence of events, possibly empty, $e_1, e_2 \ldots e_n$ such that

$$M_0 \xrightarrow{e_1} M_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} M_n = M.$$

in N.

There is an alternative characterisation of the transitions between markings induced by event occurrences:

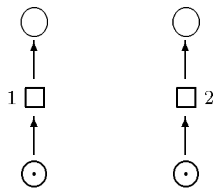
Proposition 42 *Let N be a net with markings M, M' and event e. Then*

$$M \xrightarrow{e} M' \text{ iff } (1) {}^\bullet e \subseteq M \;\&\; e^\bullet \cap (M \setminus {}^\bullet e) = \emptyset \text{ and}$$
$$(2) M' = (M \setminus {}^\bullet e) \cup e^\bullet.$$

Property (1) expresses that the event e has *concession* at the marking M, while property (2) shows that the marking resulting from the occurrence of an event at a marking is unique.

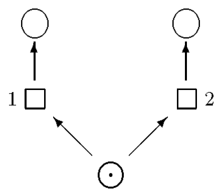
We illustrate by means of a few small examples how nets can be used to model nondeterminism and concurrency. We make use of the commonly accepted graphical notations for nets in which events are represented by squares, conditions by circles and the pre and post condition maps by directed arcs. The holding of a condition is represented by marking it by a "token"; the distribution of tokens changes as the "token game" expressed in section 9 takes place.

Example:
(1) Concurrency:

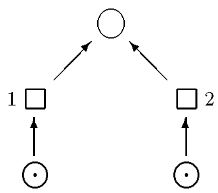


The events 1 and 2 can occur concurrently, in the sense that they both have concession and are independent in not having any pre or post conditions in common.

(2) Forwards conflict:

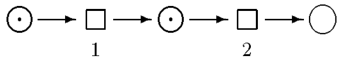


Backwards conflict:



Either one of events 1 and 2 can occur, but not both. This shows how nondeterminism can be represented in a net.

(3) Contact:



The event 2 has concession. The event 1 does not—its post condition holds—and it can only occur after 2.

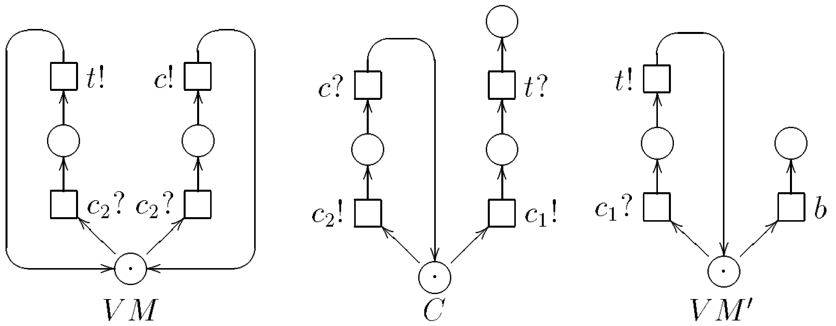
Example (3) above illustrates contact. In general, there is *contact* at a marking M when for some event e

$$^{\bullet}e \subseteq M \ \& \ e^{\bullet} \cap (M \setminus {}^{\bullet}e) \neq \emptyset.$$

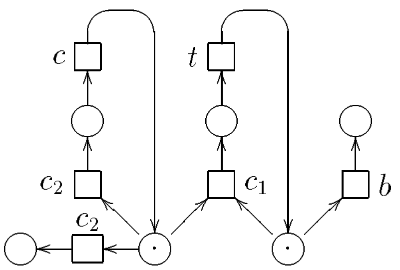
Definition: A net is said to be *safe* when contact never occurs at any reachable marking.

Many constructions on nets preserve safeness. As we shall see any net can be turned into a safe net with essentially the same behaviour—this follows from the coreflection between certain asynchronous transition systems and nets dealt with in section 10.2.2.

Example: We illustrate how one might model the customer-vending machine example of 3.3 by a net. We can represent its components by following nets, in which the events are labelled:



Their composition as SYS can be represented by the following labelled net:



The fact that the b-event can occur concurrently (or in parallel with) either of the c_2-events is reflected in the b-event and c_2-event both having concession and being independent.

As this example makes clear, what's needed are operations on nets to build up this net description (or one with essentially the same behaviour). These will appear as universal constructions in the category of labelled nets.

9.1 A category of Petri nets

As morphisms on nets we take:[7]

Definition: Let $N = (B, M_0, E, pre, post)$ and $N' = (B', M'_0, E', pre', post')$ be nets. A *morphism* $(\beta, \eta) : N \to N'$ consists of a relation $\beta \subseteq B \times B'$, such that β^{op} is a partial function $B' \to B$, and a partial function $\eta : E \to E'$ such that

$$\begin{aligned} \beta M_0 &= M'_0, \\ \beta {}^\bullet e &= {}^\bullet\eta(e) \text{ and} \\ \beta e^\bullet &= \eta(e)^\bullet. \end{aligned}$$

Thus morphisms on nets preserve initial markings and events when defined. A morphism $(\beta, \eta) : N \to N'$ expresses how occurrences of events and conditions in N induce occurrences in N'. Morphisms on nets preserve behaviour:

Proposition 43 *Let $N = (B, M_0, E, pre, post)$, $N' = (B', M'_0, E', pre', post')$ be nets. Suppose $(\beta, \eta) : N \to N'$ is a morphism of net.*

- *If $M \xrightarrow{e} M'$ in N then $\beta M \xrightarrow{\eta(e)} \beta M'$ in N'.*
- *If ${}^\bullet e_1^\bullet \cap {}^\bullet e_2^\bullet = \emptyset$ in N then ${}^\bullet\eta(e_1)^\bullet \cap {}^\bullet \eta(e_2)^\bullet = \emptyset$ in N'.*

Proof: By definition,

$$ {}^\bullet\eta(e) = \beta {}^\bullet e \text{ and } \eta(e)^\bullet = \beta e^\bullet $$

for e an event of N. Observe too that because β^{op} is a partial function, β in addition preserves intersections and set differences. These observations mean that $\beta M \xrightarrow{\eta(e)} \beta M'$ in N' follows from the assumption that $M \xrightarrow{e} M'$ in N, and that independence is preserved. $\square$

Proposition 44 *Nets and their morphisms form a category in which the composition of two morphisms $(\beta_0, \eta_0) : N_0 \to N_1$ and $(\beta_1, \eta_1) : N_1 \to N_2$ is $(\beta_1 \circ \beta_0, \eta_1 \circ \eta_0) : N_0 \to N_2$ (composition in the left component being that of relations and in the right that of partial functions).*

Definition: Let **N** be the category of nets described above.

[7] The morphisms on nets will preserve the transition-and-independence behaviour of nets while, as usual, respecting a choice of granularity fixed by the events. The rich structure of conditions on nets leaves room for variation, and another definition of morphism gives sensible results on the subclass of safe nets—see section 10.3.2.

9.2 Constructions on nets

We examine some of the more important constructions in the category of nets. There are several constructions on nets which achieve the behaviour required of a nondeterministic sum of processes. We describe a coproduct in the category of nets.

Definition: Let $N_0 = (B_0, M_0, E_0, pre_0, post_0)$ and $N_1 = (B_1, M_1, E_1, pre_1, post_1)$ be nets. Define $N_0 + N_1$ to be $(B, M, E, pre, post)$ where

$$B = M_0 \times M_1 \cup (B_0 \setminus M_0) \times_* (B_1 \setminus M_1),$$

which is associated with relations $j_0 \subseteq B_0 \times B, j_1 \subseteq B_1 \times B$ given by

$$\begin{array}{rcl} b_0 j_0 c & \Leftrightarrow & \exists b_1 \in B_1 \cup \{*\}.\ c = (b_0, b_1) \\ b_1 j_1 c & \Leftrightarrow & \exists b_0 \in B_0 \cup \{*\}.\ c = (b_0, b_1), \qquad \text{and further} \\ M & = & M_0 \times M_1, \\ E & = & E_0 \uplus E_1,\ \text{a disjoint union associated with injections} \\ & & in_0 : E_0 \to E_1, in_1 : E_1 \to E,\ \text{and finally} \\ pre(e) & = & j_0 \circ pre_0(e_0) \text{ and} \\ post(e) & = & j_0 \circ post_0(e_0) \text{ if } e = in_0(e_0),\ \text{and} \\ pre(e) & = & j_1 \circ pre_1(e_1) \text{ and} \\ post(e) & = & j_1 \circ post_1(e_1) \text{ if } e = in_1(e_1). \end{array}$$

The only peculiarity in this definition is the way in which the conditions are built. However, note that the relation β in any morphism

$$(\beta, \eta) : N \to N'$$

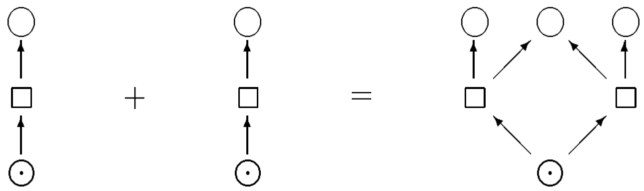
of nets N, N', with conditions B, B' and initial markings M, M' respectively, corresponds to a pair of functions

$$\begin{array}{rl} \beta^{op} : & (B' \setminus M') \to (B \setminus M) \text{ in } \mathbf{Set}_*, \\ \beta^{op} : & M' \to M \text{ in } \mathbf{Set}. \end{array}$$

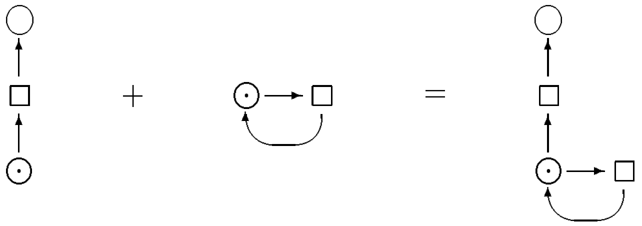
Thus it is to be expected that the conditions of a coproduct of nets correspond to products in $\mathbf{Set}_* \times \mathbf{Set}$. This remark handles the only obstacle in the proof of:

Proposition 45 *The construction $N_0 + N_1$ above is a coproduct in the category of nets $\mathbf{N}$ with injections $(j_0, in_0) : N_0 \to N_0 + N_1, (j_1, in_1) : N_1 \to N_0 + N_1$.*

Example:
(1)



(2)



In general the coproduct of nets can behave strangely, and allow a mix of behaviours from the two component nets. However, in the case where the component nets are safe, as they are in the example above, their coproduct is safe too and has a behaviour which can be described in terms of that of the components using the injection morphisms.

Lemma 46 *Suppose N_0, N_1 are safe nets with initial markings M_0, M_1 respectively. Then their coproduct $N_0 + N_1$ is safe. Moreover:*

(1a) *Two events $in_0(e_0), in_0(e'_0)$ are independent in $N_0 + N_1$ iff events e_0, e'_0 are independent in N_0. Similarly, two events $in_1(e_1), in_1(e'_1)$ are independent in $N_0 + N_1$ iff events e_1, e'_1 are independent in N_1.*

(1b) *Two events $in_0(e_0), in_1(e_1)$ are independent in $N_0 + N_1$ iff*

$$\begin{array}{l} {}^\bullet e_0{}^\bullet \subseteq M_0 \textit{ in } N_0 \;\;\& \; {}^\bullet e_1{}^\bullet \cap M_1 = \emptyset \textit{ in } N_1, \textit{ or} \\ {}^\bullet e_1{}^\bullet \subseteq M_1 \textit{ in } N_1 \;\;\& \; {}^\bullet e_0{}^\bullet \cap M_0 = \emptyset \textit{ in } N_0. \end{array}$$

(2) *X is reachable & $X \xrightarrow{e} X'$ in $N_0 + N_1$ iff*

$$\begin{array}{l} \exists e_0, \textit{reachable } X_0, X'_0. \\ e = in_0(e_0) \;\&\; X_0 \xrightarrow{e_0} X'_0 \text{ in } N_0 \;\&\; X = j_0 X_0 \;\&\; X' = j_0 X'_0, \;\; \textit{or} \\ \exists e_1, \textit{reachable } X_1, X'_1. \\ e = in_1(e_1) \;\&\; X_1 \xrightarrow{e_1} X'_1 \text{ in } N_1 \;\&\; X = j_1 X_1 \;\&\; X' = j_1 X'_1. \end{array}$$

Proof: (1a) is obvious, and (1b) follows from the way the conditions of the coproduct are constructed. The "if" direction of (2) follows as the injections are morphisms. The "only if" direction follows by showing: if X_0 is a reachable marking of N_0 and $j_0 X_0 \xrightarrow{e} X'$ in $N_0 + N_1$ then either

(a) $e = in_1(e_1)$ & $X_0 = M_0$ & $X' = j_1 X'_1$ & $M_1 \xrightarrow{e_1} X'_1$, for some event e_1 and marking X'_1 of N_1 , or

(b) $e = in_0(e_0)$ & $X' = j_0 X'_0$ & $X_0 \xrightarrow{e_0} X'_0$ in N_0, for some event e_0 and marking X'_0 of N_0.

To show this, assume $j_0 X_0 \xrightarrow{e} X'$ in $N_0 + N_1$ where X_0 is a reachable marking of N_0. Consider first the case where $e = in_1(e_1)$. Because $in_1(e_1)$ has concession at $j_0 X_0$

$$^\bullet in_1(e_1) \subseteq j_0 X_0$$

from which we see

$$^\bullet e_1 \subseteq M_1$$

—otherwise $in_1(e_1)$ would have a precondition of the form $(*, b_1)$ which cannot be in the image $j_0 X_0$ of the marking X_0 of N_0. Because, by assumption, we have some $b_1 \in ^\bullet e_1$ we see that

$$M_0 \times \{b_1\} \subseteq \ ^\bullet in_1(e_1)$$

Because we now have

$$M_0 \times \{b_1\} \subseteq j_0 X_0$$

it follows that $M_0 \subseteq X_0$. But N_0 is safe and X_0 is assumed reachable from M_0, so we must have $M_0 = X_0$—otherwise a repetition of the same "token game" which led from M_0 to X_0, but this time starting from X_0, would lead to contact. Letting X'_1 be the marking such that $M_1 \xrightarrow{e_1} X'_1$ we calculate

$$\begin{aligned} X' =& (j_0 X_0 \setminus^\bullet in_1(e_1)) \cup in_1(e_1)^\bullet \\ =& (M_0 \times M_1 \setminus^\bullet in_1(e_1)) \cup in_1(e_1)^\bullet \\ =& (j_1 M_1 \setminus j_1 \ ^\bullet e_1) \cup j_1 e_1^\bullet \\ =& j_1((M_1 \setminus^\bullet e_1) \cup e_1^\bullet) \\ =& j_1 X'_1. \end{aligned}$$

This establishes (a) in the case where $e = in_1(e_1)$ In the other case, where $e = in_0(e_0)$, a similar but easier argument establishes (b). An analogous result holds for N_1 in place of N_0. The "only if" direction of (2) now follows.

Suppose $N_0 + N_1$ were not safe. Then

$$^\bullet e \subseteq X \ \& \ e^\bullet \cap (X \setminus ^\bullet e) \neq \emptyset$$

for some reachable marking X and event e of N_0+N_1. Suppose $e = in_0(e_0)$. Then by the results above, without loss of generality, we can suppose that $X = j_0X_0$ for some reachable marking X_0 of N_0. By the definition of the pre and post conditions of events of $N_0 + N_1$ we then obtain

$$^\bullet e_0 \subseteq X_0 \ \& \ e_0{}^\bullet \cap (X_0 \setminus {}^\bullet e_0) \neq \emptyset,$$

contradicting the assumption that N_0 is safe. $\square$

The product of nets and its behaviour are more straightforward, and as is to be expected correspond to a synchronisation operation on nets.

Definition: Let $N_0 = (B_0, M_0, E_0, pre_0, post_0)$ and $N_1 = (B_1, M_1, E_1, pre_1, post_1)$ be nets. Their *product* $N_0 \times N_1 = (B, E, M, pre, post)$; it has events

$$E = E_0 \times_* E_1,$$

the product in $\mathbf{Set}_*$ with projections $\pi_0 : E \to_* E_0$ and $\pi_1 : E \to_* E_1$. Its conditions have the form $B = B_0 \uplus B_1$ the disjoint union of B_0 and B_1. Define ρ_0 to be the opposite relation to the injection $\rho_0{}^{op} : B_0 \to B$. Define ρ_1 similarly. Take $M = \rho_0{}^{op}M_0 + \rho_1{}^{op}M_1$ as the initial marking of the product. Define the pre and postconditions of an event e in the product in terms of its pre and postconditions in the components by

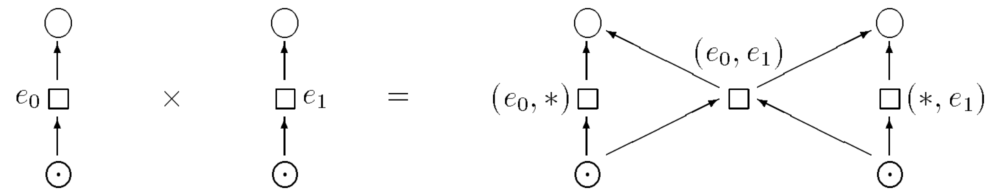
$$\begin{aligned} pre(e) &= \rho_0^{op}[pre_0(\pi_0(e))] + \rho_1^{op}[pre_1(\pi_1(e))] \\ post(e) &= \rho_0^{op}[post_0(\pi_0(e))] + \rho_1^{op}[post_1(\pi_1(e))]. \end{aligned}$$

Proposition 47 *The product $N_0 \times N_1$, with morphisms (ρ_0, π_0) and (ρ_1, π_1), is a product in the category of Petri nets.*

Proposition 48 *The behaviour of a product of nets $N_0 \times N_1$ is related to the behaviour of its components N_0 and N_1 by*

$$M \xrightarrow{e} M' \ \textit{in}\ N_0 \times N_1 \quad \textit{iff} \quad (\rho_0 M \xrightarrow{\pi_0(e)} \rho_0 M' \ \textit{in}\ N_0 \ \& \ \rho_1 M \xrightarrow{\pi_1(e)} \rho_1 M' \ \textit{in}\ N_1).$$

Example: The product of two nets:



10 Asynchronous transition systems

Asynchronous transition systems deserve to be better known as a model of parallel computation. They were introduced independently by Bednarczyk in [5] and Shields in [84]. The idea on which they are based is simple enough: extend transition systems by, in addition, specifying which transitions are independent of which. More accurately, transitions are to be thought of as occurrences of events which bear a relation of independence. This interpretation is supported by axioms which essentially generalise those from Mazurkiewicz trace languages.

Definition: An *asynchronous transition system* consists of $(S, i, E, I, \mathit{Tran})$ where (S, i, E, Tran) is a transition system, $I \subseteq E^2$, the *independence relation* is an irreflexive, symmetric relation on the set E of *events* such that

(1) $e \in E \Rightarrow \exists s, s' \in S.\ (s, e, s') \in \mathit{Tran}$

(2) $(s, e, s') \in \mathit{Tran} \ \&\ (s, e, s'') \in \mathit{Tran} \Rightarrow s' = s''$

(3) $e_1 I e_2 \ \&\ (s, e_1, s_1) \in \mathit{Tran} \ \&\ (s, e_2, s_2) \in \mathit{Tran}$
$\quad \Rightarrow \exists u.\ (s_1, e_2, u) \in \mathit{Tran} \ \&\ (s_2, e_1, u) \in \mathit{Tran}$

(4) $e_1 I e_2 \ \&\ (s, e_1, s_1) \in \mathit{Tran} \ \&\ (s_1, e_2, u) \in \mathit{Tran}$
$\quad \Rightarrow \exists s_2.\ (s, e_2, s_2) \in \mathit{Tran} \ \&\ (s_2, e_1, u) \in \mathit{Tran}$

Axiom (1) says every event appears as a transition, and axiom (2) that the occurrence of an event at a state leads to a unique state. Axioms (3) and (4) express properties of independence: if two events can occur independently from a common state then they should be able to occur together and in so doing reach a common state (3); if two independent events can occur one immediately after the other then they should be able to occur with their order interchanged (4). Both situations lead to an "independence square" associated with the independence $e_1 I e_2$:

$$\begin{tikzcd}
 & u & \\
s_1 \arrow[ur, "e_2"] & & s_2 \arrow[ul, "e_1"'] \\
 & s \arrow[ul, "e_1"] \arrow[ur, "e_2"'] &
\end{tikzcd}$$

Axiom (3) corresponds to the coherence axiom on Mazurkiewicz trace languages, and, as there, a great deal of the theory can be developed without it.[8]

[8] Without axiom (3) asynchronous transition systems generate Mazurkiewicz trace languages, but without the coherence axiom, and unfold to more general event structures than those with a *binary* conflict.

Example: We return to the example of 3.3. The transition system representing SYS, in section 3.3 cannot be the underlying transition system of an asynchronous transition system with events identified with labels—there are, for example distinct c_2 transitions from the initial state. If however we distinguish between the two ways in which c_2 can occur, and take the set of events to be

$$\{b, c, c_1, c_2', c_2'', t\}$$

and explicitly assert the independencies

$$b\ I\ c_2', \text{and } b\ I\ c_2''$$

we can obtain an asynchronous transition system with transitions

$$\begin{tikzcd}
 & & & \bullet \arrow[r, "b"] \arrow[d, shift right, "c"'] & \bullet \arrow[d, shift left, "c"] \\
\bullet & \bullet \arrow[l, "b"'] & \bullet \arrow[l, "t"'] & \odot \arrow[l, "c_1"'] \arrow[u, shift right, "c_2'"'] \arrow[r, "b"] \arrow[d, "c_2''"] & \bullet \arrow[u, shift left, "c_2'"] \arrow[d, "c_2''"'] \\
 & & & \bullet \arrow[r, "b"] & \bullet
\end{tikzcd}$$

The parallelism is caught by the independence squares (two upper and one lower).

Morphisms between asynchronous transition systems are morphisms between their underlying transition systems which preserve the additional relations of independence.

Definition: Let $T = (S, i, E, I, \mathit{Tran})$ and $T' = (S', i', E', I', \mathit{Tran}')$ be asynchronous transition systems. A *morphism* $T \to T'$ is a morphism of transition systems

$$(\sigma, \eta) : (S, i, E, \mathit{Tran}) \to (S', i', E', \mathit{Tran}')$$

such that

$$e_1 I e_2 \ \&\ \eta(e_1), \eta(e_2) \text{ both defined } \Rightarrow \eta(e_1) I' \eta(e_2).$$

Morphisms of asynchronous transition systems compose as morphisms between their underlying transition systems, and are readily seen to form a category.

Definition: Write $\mathbf{A}$ for the category of asynchronous transition systems.

The category $\mathbf{A}$ has categorical constructions which essentially generalise those of transition systems and Mazurkiewicz traces. Here are the product and coproduct constructions for the category $\mathbf{A}$:

Definition: Assume asynchronous transition systems $T_0 = (S_0, i_0, E_0, I_0, \mathit{Tran}_0)$ and $T_1 = (S_1, i_1, E_1, I_1, \mathit{Tran}_1)$. Their *product* $T_0 \times T_1$ is $(S, i, E, I, \mathit{Tran})$ where (S, i, E, Tran) is the product of transition systems $(S_0, i_0, E_0, \mathit{Tran}_0)$ and $(S_1, i_1, E_1, \mathit{Tran}_1)$ with projections (ρ_0, π_0) and (ρ_1, π_1), and the independence relation I is given by

$$\begin{aligned} aIb \Leftrightarrow & (\pi_0(a), \pi_0(b) \text{ defined } \Rightarrow \pi_0(a) I_0 \pi_0(b)) \ \& \\ & (\pi_1(a), \pi_1(b) \text{ defined } \Rightarrow \pi_1(a) I_1 \pi_1(b)). \end{aligned}$$

Definition: Assume asynchronous transition systems $T_0 = (S_0, i_0, E_0, I_0, \mathit{Tran}_0)$ and $T_1 = (S_1, i_1, E_1, I_1, \mathit{Tran}_1)$. Their *coproduct* $T_0 + T_1$ is $(S, i, E, I, \mathit{Tran})$ where (S, i, E, Tran) is the coproduct of transition systems $(S_0, i_0, E_0, \mathit{Tran}_0)$ and $(S_1, i_1, E_1, \mathit{Tran}_1)$ with injections (in_0, j_0) and (in_1, j_1), and the independence relation I is given by

$$\begin{aligned} aIb \Leftrightarrow & (\exists a_0, b_0.\ a = j_0(a_0) \ \&\ b = j_0(b_0) \ \&\ a_0 I_0 b_0) \text{ or} \\ & (\exists a_1, b_1.\ a = j_1(a_1) \ \&\ b = j_1(b_1) \ \&\ a_1 I_1 b_1). \end{aligned}$$

10.1 Asynchronous transition systems and trace languages

That asynchronous transition systems generalise trace languages is backed up by a straightforward coreflection between categories of trace languages and asynchronous transition systems. To obtain the adjunction we need to restrict trace languages to those where every element of the alphabet occurs in some trace (this matches property (1) required by the definition of asynchronous transition systems).

Definition: Define $\mathbf{TL}_0$ to be the full subcategory of trace languages (M, E, I) satisfying

$$\forall e \in E \exists s.\ se \in M.$$

A trace language forms an asynchronous transition system in which the states are traces.

Definition: Let (M, E, I) be a trace language in $\mathbf{TL}_0$, with trace equivalence $\approx$. Define $\mathit{tla}(M, E, I) = (S, i, E, I, \mathit{Tran})$ where

$$\begin{aligned} S &= M/\!\approx \ \text{ with } i = \{\epsilon\}_\approx \\ (t, e, t') \in \mathit{Tran} &\Leftrightarrow \exists s, se \in M.\ t = \{s\}_\approx \ \&\ t' = \{se\}_\approx \end{aligned}$$

Let $\eta : (M, E, I) \to (M', E', I')$ be a morphism of trace languages. Define $\mathit{tla}(\eta) = (\sigma, \eta)$ where

$$\sigma(\{s\}_\approx) = \{\hat{\eta}(s)\}_\approx.$$

(Note this is well-defined because morphisms between trace languages respect $\approx$—this follows directly from proposition 17.)

Proposition 49 *The operation tla is a functor* $\mathbf{TL}_0 \to \mathbf{A}$.

An asynchronous transition system determines a trace language:

Definition: Let $T = (S, i, E, I, \mathit{Tran})$ be an asynchronous transition system. Define $\mathit{atl}(T) = (\mathit{Seq}, E, I)$ where Seq consists of all strings of events, possibly empty, $e_1 e_2 \dots e_n$ for which there are transitions

$$(i, e_1, s_1), (s_1, e_2, s_2), \dots, (s_{n-1}, e_n, s_n) \in \mathit{Tran}$$

Let $(\sigma, \eta) : T \to T'$ be a morphism of asynchronous transition systems. Define $\mathit{atl}(\sigma, \eta) = \eta$.

Proposition 50 *The operation atl is a functor* $\mathbf{A} \to \mathbf{TL}_0$.

In fact, the functors tla, atl form a coreflection:

Theorem 51 *The functor* $\mathit{tla} : \mathbf{TL}_0 \to \mathbf{A}$ *is left adjoint to* $\mathit{atl} : \mathbf{A} \to \mathbf{TL}_0$.

Let $L = (M, E, I)$ *be a trace language. Then* $\mathit{atl} \circ \mathit{tla}\ (M, E, I) = (M, E, I)$ *and the unit of the adjunction at* (M, E, I) *is the identity* $1_E : (M, E, I) \to \mathit{atl} \circ \mathit{tla}(M, E, I)$.

Let T *be an asynchronous transition system, with events* E. *Then* $(\sigma, 1_E) : \mathit{tla} \circ \mathit{atl}(T) \to T$ *is the counit of the adjunction at* T, *where* $\sigma(t)$, *for a trace* $t = \{e_1 e_2 \dots e_n\}_{\approx}$, *equals the unique state* s *for which* $i \xrightarrow{e_1 e_2 \dots e_n} s$.

Proof: Let $L = (M, E, I)$ be a trace language in $\mathbf{TL}_0$ and $T = (S, i, E', I', \mathit{Tran}')$ be an asynchronous system. Given a morphism of trace languages

$$\eta : L \to \mathit{atl}(T)$$

there is a unique morphism of asynchronous transition systems

$$(\sigma, \eta) : \mathit{tla}(L) \to T$$

—the function σ must act so $\sigma(t)$, on a trace $t = \{e_1 e_2 \dots e_n\}_{\approx}$, equals the unique state s_n for which there are transitions, possibly idle,

$$(i, \eta(e_1), s_1), (s_1, \eta(e_2), s_2), \dots, (s_{n-1}, \eta(e_n), s_n)$$

in T. That this is well-defined follows from T satisfying axiom 4 in the definition of asynchronous transition systems. The stated coreflection, and the form of the counit, follow. $\square$

The coreflection does not extend to an adjunction from **TL** to **A**—$\mathbf{TL}_0$ is a reflective and not a coreflective subcategory of **TL**.

We note that a coreflection between event structures and asynchronous transition systems follows by composing the coreflections between event structures and trace languages and that between trace languages and asynchronous transition systems. It is easy to see that the coreflection $\mathbf{E} \hookrightarrow \mathbf{TL}$ restricts to a coreflection to $\mathbf{E} \hookrightarrow \mathbf{TL}_0$. The left adjoint of the resulting coreflection, is the composition

$$\mathbf{E} \xrightarrow{etl} \mathbf{TL}_0 \xrightarrow{tla} \mathbf{A}.$$

A left adjoint of the coreflection can however be constructed more directly. The composition $tla \circ etl$ is naturally isomorphic to the functor yielding an asynchronous transition system directly out of the configurations of the event structure, as is described in the next proposition.

Proposition 52 *For $ES = (E, \leq, \#)$ an event structure, define*

$$\Gamma(ES) = (\mathcal{D}^0(ES), \emptyset, E, co, Tran)$$

where the transitions between configurations, Tran, consist of (x, e, x') where $e \notin x$ & $x' = x \cup \{e\}$. For $\eta : ES \to ES'$ a morphism of event structures, define $\Gamma(\eta) = (\sigma, \eta)$ where $\sigma(x) = \eta x$, for x a configuration of ES. This defines a functor $\Gamma : \mathbf{E} \to \mathbf{A}$. Moreover, Γ is naturally isomorphic to $tla \circ etl$.

Proof: It is easy to check that Γ is a functor. The representation theorem 33, and its consequence, proposition 34, yield a morphism

$$(Ev^{-1}, \lambda_T) : \Gamma \circ tle(T) \to tla(T),$$

of asynchronous transition systems, which can be checked to be natural in T. Letting T be the trace language $etl(ES)$, of an event structure ES, we obtain a morphism

$$(Ev^{-1}, \lambda_{etl(ES)}) : \Gamma \circ tle \circ etl(ES) \to tla \circ etl(ES),$$

natural in ES. The coreflection 39 ensures that the counit at $etl(ES)$

$$\lambda_{etl(ES)} : etl \circ tle \circ etl(ES) \to etl(ES)$$

is an isomorphism. This makes the function $\lambda_{etl(ES)}$ a bijection, which together with the bijection Ev given by the representation theorem 33, ensures $(Ev^{-1}, \lambda_{etl(ES)})$ is an isomorphism, necessarily natural in ES. It composes with the natural isomorphism $\Gamma(\eta_{ES}) : \Gamma(ES) \to \Gamma \circ tle \circ etl(ES)$, where $\eta_{ES} : ES \to tle \circ etl(ES)$ is the unit of the coreflection at ES, to give the required natural isomorphism. $\square$

10.2 Asynchronous transition systems and nets

10.2.1 An adjunction

There is an adjunction between the categories $\mathbf{A}$ and $\mathbf{N}$. First, we note there is an obvious functor from nets to asynchronous transition systems.

Definition: Let $N = (B, M_0, E, {}^\bullet(\ ), (\ )^\bullet)$ be a net. Define $na(N) = (S, i, E, I, \mathit{Tran})$ where

$$S = \mathcal{P}ow(B) \text{ with } i = M_0,$$
$$e_1 I e_2 \Leftrightarrow {}^\bullet e_1^\bullet \cap {}^\bullet e_2^\bullet = \emptyset,$$
$$(M, e, M') \in \mathit{Tran} \Leftrightarrow M \xrightarrow{e} M' \text{ in } N, \text{ for } M, M' \in \mathcal{P}ow(B).$$

Let $(\beta, \eta) : N \to N'$ be a morphism of nets. Define

$$na(\beta, \eta) = (\sigma, \eta)$$

where $\sigma(M) = \beta M$, for any $M \in \mathcal{P}ow(B)$.

Proposition 53 *na is a functor* $\mathbf{N} \to \mathbf{A}$.

Proof: Letting N be a net, it is easily checked that $na(N)$ is an asynchronous transition system: properties (1) and (2) of definition 10 are obvious while properties (3) and (4) follow directly from the interpretation of independence of events e_1, e_2 as ${}^\bullet e_1^\bullet \cap {}^\bullet e_2^\bullet = \emptyset$. Letting $(\beta, \eta) : N \to N'$ be a morphism of nets, proposition 43 yields that $na(\beta, \eta)$ is a morphism $na(N) \to na(N')$. Clearly na preserves composition and identities. $\square$

As a preparation for the definition of a functor from asynchronous transition systems to nets we examine how a condition of a net N can be viewed as a subset of states and transitions of the asynchronous transition system $na(N)$. Intuitively the *extent* $|b|$ of a condition b of a net is to consist of those markings and transitions at which b holds uninterruptedly. In fact, for simplicity, the extent $|b|$ of a condition b is taken to be a subset of Tran_*, the transitions (M, e, M') and idle transitions $(M, *, M)$ of $na(N)$; the idle transitions $(M, *, M)$ play the role of markings M.

Definition: Let b be a condition of a net N. Let Tran be the transition relation of $na(N)$. Define the *extent* of b to be

$$|b| = \{(M, e, M') \in \mathit{Tran}_* \mid b \in M \ \&\ b \in M' \ \&\ b \notin {}^\bullet e^\bullet\}.$$

Not all subsets of transitions $Tran_*$ of a net N are extents of conditions of N. For example, if $(M, e, M') \notin |b|$ and $(M', *, M') \in |b|$ for a transition $M \xrightarrow{e} M'$ in N this means the transition starts the holding of b. But then $b \in e^\bullet$ so any other transition $P \xrightarrow{e} P'$ must also start the holding of b. Of course, a condition cannot be started or ended by two independent events because, by definition, they can have no pre- or postcondition in common. These considerations motivate the following definition of condition of a general asynchronous transition system.

Definition: Let $T = (S, i, E, I, Tran)$ be an asynchronous transition system. Its *conditions* are nonempty subsets $b \subseteq Tran_*$ such that

(1) $(s, e, s') \in b \Rightarrow (s, *, s) \in b \;\&\; (s', *, s') \in b$

(2) (i) $(s, e, s') \in^\bullet b \;\&\; (u, e, u') \in Tran \Rightarrow (u, e, u') \in^\bullet b$

(ii) $(s, e, s') \in b^\bullet \;\&\; (u, e, u') \in Tran \Rightarrow (u, e, u') \in b^\bullet$

where for $(s, e, s') \in Tran$ we define

$(s, e, s') \in^\bullet b \Leftrightarrow (s, e, s') \notin b \;\&\; (s', *, s') \in b,$
$(s, e, s') \in b^\bullet \Leftrightarrow (s, *, s) \in b \;\&\; (s, e, s') \notin b$ and
${}^\bullet b^\bullet = {}^\bullet b \cup b^\bullet.$

(3) $(s, e_1, s') \in {}^\bullet b^\bullet \;\&\; (u, e_2, u') \in {}^\bullet b^\bullet \Rightarrow \neg e_1 I e_2.$

Let B be the set of conditions of T. For $e \in E_*$, define

$$\begin{aligned} e^\bullet &= \{b \in B \mid \exists s, s'. \, (s, e, s') \in^\bullet b\}, \\ {}^\bullet e &= \{b \in B \mid \exists s, s'. \, (s, e, s') \in b^\bullet\}, \text{ and} \\ {}^\bullet e^\bullet &= {}^\bullet e \cup e^\bullet. \end{aligned}$$

(Note that ${}^\bullet *^\bullet = \emptyset$.)
Further, for $s \in S$, define $M(s) = \{b \in B \mid (s, *, s) \in b\}$.

As an exercise, we check that the extent of a condition of a net is indeed a condition of its asynchronous transition system.

Lemma 54 *Let N be a net with a condition b. Its extent $|b|$ is a condition of $na(N)$. Moreover,*

(I) $(M, e, M') \in^\bullet |b| \Leftrightarrow b \in e^\bullet$

(II) $(M, e, M') \in |b|^\bullet \Leftrightarrow b \in^\bullet e.$

whenever $M \xrightarrow{e} M'$ in N.

Proof: We prove (I) (the proof of (II) is similar):

$$\begin{array}{rcl} (M,e,M') \in^\bullet |b| & \Leftrightarrow & (M,e,M') \notin |b| \ \& \ (M',*,M') \in |b| \\ & \Leftrightarrow & \neg(b \in M \ \& \ b \in M' \ \& \ b \notin^\bullet e^\bullet) \ \& \ b \in M' \\ & \Leftrightarrow & (b \notin M \ \& \ b \in M') \text{ or } (b \in^\bullet e^\bullet \ \& \ b \in M') \\ & \Leftrightarrow & b \in e^\bullet, \text{ as } M \xrightarrow{e} M'. \end{array}$$

Using (I) and (II), it is easy to check that $|b|$ is a condition of $na(N)$. First we note $|b|$ is nonempty because it contains for instance $(\{b\},*,\{b\})$. We quickly run through the axioms required by definition 10.2.1:

(1) If $(M,e,M') \in |b|$ then $b \in M$ and $b \in M'$ whence $(M,*,M),(M',*,M') \in |b|$.

(2) (i) If $(M,e,M') \in^\bullet |b|$ then $b \in e^\bullet$, by (I) " $\Rightarrow$". Hence, if $P \xrightarrow{e} P'$ by (I) "$\Leftarrow$" we obtain $(P,e,P') \in^\bullet |b|$. The proof of (2)(ii) is similar.

(3) (i) If $(M,e_1,M'),(P,e_2,P') \in^\bullet |b|$ then $b \in e_1^\bullet$ and $b \in e_2^\bullet$, by (I) applied twice. Hence $\neg e_1 I e_2$. □

Definition: Let $(\sigma,\eta): T \to T'$ be a morphism between asynchronous transition systems $T = (S,i,E,I,\mathit{Tran})$ and $T' = (S',i',E',I',\mathit{Tran}')$. For $b \subseteq \mathit{Tran}'_*$, define

$$(\sigma,\eta)^{-1}b = \{(s,e,s') \in \mathit{Tran}_* \mid (\sigma(s),\eta(e),\sigma(s')) \in b\}$$

Lemma 55 *Let $(\sigma,\eta): T \to T'$ be a morphism between asynchronous transition systems. Let b be a condition of T'. Then $(\sigma,\eta)^{-1}b$ is a condition of T provided it is nonempty. Furthermore,*

$$\begin{array}{lrcl} (1) & (\sigma,\eta)^{-1}b \in^\bullet e & \Leftrightarrow & b \in^\bullet \eta(e) \\ (2) & (\sigma,\eta)^{-1}b \in e^\bullet & \Leftrightarrow & b \in \eta(e)^\bullet \end{array}$$

for any event e of T.

Proof: We show (1), assuming $b \subseteq \mathit{Tran}'_*$ and an event e of T. Observe

$$\begin{array}{rcl} (\sigma,\eta)^{-1}b \in^\bullet e & \Leftrightarrow & (s,e,s') \in (\sigma,\eta)^{-1}b^\bullet, \text{ for some states } s,s' \\ & \Leftrightarrow & (s,*,s) \in (\sigma,\eta)^{-1}b \ \& \ (s,e,s') \notin (\sigma,\eta)^{-1}b \\ & \Leftrightarrow & (\sigma(s),*,\sigma(s)) \in b \ \& \ (\sigma(s),\eta(e),\sigma(s')) \notin b \\ & \Leftrightarrow & (\sigma(s),\eta(e),\sigma(s')) \in b^\bullet \\ & \Leftrightarrow & b \in^\bullet \eta(e) \end{array}$$

The proof of (2) is similar. That $(\sigma,\eta)^{-1}b$ is a condition of T, if nonempty, follows straightforwardly from the assumption that b is a condition. □

Definition: Let $T = (S, i, E, I, Tran)$ be an asynchronous transition system. Define $an(T) = (B, M_0, E, pre, post)$ by taking B to be the set of conditions of T, $M_0 = M(i)$, with pre and post condition maps given by the corresponding operations in T, *i.e.* $pre(e) =^{\bullet} e$ and $post(e) = e^{\bullet}$ in T. Let $(\sigma, \eta) : T \to T'$ be a morphism of asynchronous transition systems. Define $an(\sigma, \eta) = (\beta, \eta)$ where for conditions b of T and b' of T' we take

$$b \beta b' \text{ iff } b = (\sigma, \eta)^{-1} b'.$$

(Note that because of lemma 55,

$$b \beta b' \text{ iff } \emptyset \neq b = (\sigma, \eta)^{-1} b'$$

where we only assume b' is a condition of T'.)

The verification that $an(T)$ is indeed a net involves demonstrating that every event has at least one pre and post condition. This follows from the following lemma which indicates how rich an asynchronous transition system is in conditions (it says an arbitrary pairwise-dependent set of events can be made to be both the starting and ending events of a single condition):

Lemma 56 *Let $T = (S, i, E, I, Tran)$ be an asynchronous transition system. Suppose X is a nonempty subset of E such that*

$$e_1, e_2 \in X \Rightarrow \neg e_1 I e_2.$$

Then, there is a condition b of T such that

$$X = \{e \mid b \in e^{\bullet}\} \ \& \ X = \{e \mid b \in {}^{\bullet}e\}.$$

Proof: Define

$$b = \{(s, e, s') \in Tran_* \mid e \notin X\}.$$

It is simply checked that b is a condition with beginning and ending events X. $\square$

Lemma 57 *Let $T = (S, i, E, I, Tran)$ be an asynchronous transition system. Then $an(T)$ is a net. Moreover,*

$$e_1 I e_2 \Leftrightarrow {}^{\bullet} e_1^{\bullet} \cap {}^{\bullet} e_2^{\bullet} = \emptyset,$$

and

$$(s, e, s') \in Tran \Rightarrow M(s) \xrightarrow{e} M(s') \text{ in } an(T).$$

Proof: For $an(T)$ to be a net it is required that its initial marking, and pre and post conditions of events be nonempty. However, taking $b = Tran_*$ yields a condition in the initial marking, while for an event e, letting X be $\{e\}$ in lemma 56 produces a pre and post condition of e.

If $e_1 I e_2$ then axiom (3) on conditions (definition 10.2.1) ensures ${}^\bullet e_1^\bullet \cap {}^\bullet e_2^\bullet = \emptyset$. Conversely, by lemma 56, if $\neg(e_1 I e_2)$ we can obtain a condition in ${}^\bullet e_1{}^\bullet \cap {}^\bullet e_2{}^\bullet$.

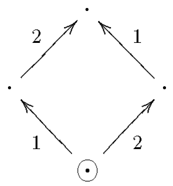
Suppose $(s, e, s') \in Tran$. Then, letting B be the set of conditions of T,

$$\begin{aligned}
{}^\bullet e &= \{b \in B \mid (s,*,s) \in b \;\&\; (s,e,s') \notin b\} \subseteq M(s), \\
e^\bullet &= \{b \in B \mid (s,e,s') \notin b \;\&\; (s',*,s') \in b\} \subseteq M(s'), \text{ and} \\
M(s) \setminus {}^\bullet e &= \{b \in B \mid (s,*,s) \in b\} \setminus \{b \in B \mid (s,*,s) \in b \;\&\; (s,e,s') \notin b\} \\
&= \{b \in B \mid (s,e,s') \in b\} \\
&= \{b \in B \mid (s',*,s') \in b\} \setminus \{b \in B \mid (s,e,s') \notin b \;\&\; (s',*,s') \in b\} \\
&= M(s') \setminus e^\bullet.
\end{aligned}$$

Thus $M(s) \xrightarrow{e} M(s')$. $\square$

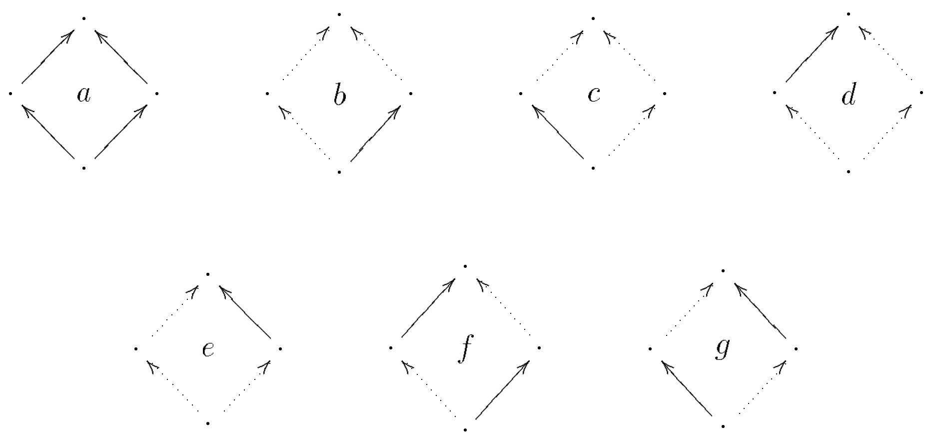
We illustrate how a net is produced from an asynchronous transition system.

Example: Consider the following asynchronous transition system T with two :

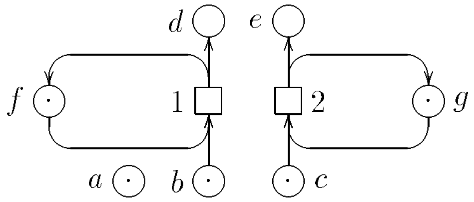


1 I 2

It has these conditions, where those transitions in the condition are represented by solid arrows:



Consequently the asynchronous transition system T yields this net $an(T)$:



Lemma 58 *an is a functor* $\mathbf{A} \to \mathbf{N}$.

Proof: The only difficulty comes in showing the well-definedness of an on morphisms. Let $(\sigma, \eta) : T \to T'$ be a morphism of asynchronous transition systems $T = (S, i, E, I, \mathit{Tran})$, $T' = (S', i', E', I', \mathit{Tran}')$. We require that $an(\sigma, \eta) =_{def} (\beta, \eta)$ is a morphism of nets $an(T) \to an(T')$. Let $an(T) = (B, M_0, E, pre, post)$, $an(T') = (B', M_0', E', pre', post')$. We see β preserves initial markings by arguing:

$$\begin{aligned} b' \in M_0' \quad &\Leftrightarrow \quad (i', *, i') \in b' \\ &\Leftrightarrow \quad (\sigma(i), *, \sigma(i)) \in b' \\ &\Leftrightarrow \quad (i, *, i) \in (\sigma, \eta)^{-1} b' \\ &\Leftrightarrow \quad \beta^{op}(b') \in M_0. \end{aligned}$$

The fact that $\beta e^\bullet = ^\bullet \eta(e)$ and $\beta^\bullet e = \eta(e)^\bullet$ follows directly from (1) and (2) of lemma 55. ☐

In fact, an is left adjoint to na. Before proving this we explore the unit and counit of the adjunction. The unit of the adjunction:

Lemma 59 *Let T be an asynchronous system. Defining $\sigma_0(s) = M(s)$ for s a state of T and letting 1_E be the identity on the events of T, we obtain a morphism of asynchronous transition systems*

$$(\sigma_0, 1_E) : T \to na \circ an(T).$$

Proof: That $(\sigma_0, 1_E)$ is a morphism follows directly from lemma 57. ☐

The counit:

Lemma 60 *Let $N = (B, M_0, E, ^\bullet(\ ), (\ )^\bullet)$ be a net. Let Tran be the transitions of $na(N)$. For $b \in B$ and $c \subseteq \mathit{Tran}_*$, taking*

$$c\beta_0 b \Leftrightarrow_{def} c = |b|$$

defines a relation between conditions of $na(N)$ and B, such that

$$(\beta_0, 1_E) : an \circ na(N) \to N$$

is a morphism of nets.

Proof: By lemma 54, $|b|$ is a condition of $na(N)$ if b is a condition of N. This ensures that β_0 is a relation between the conditions of $na(N)$ and B. We should check $(\beta_0, 1_E) : an \circ na(N) \to N$ is a morphism of nets. Let M'_0 be the initial marking of $an \circ na(N)$: We see for any $b \in B$ that

$$\begin{aligned} \beta_0^{op}(b) \in M'_0 \quad &\Leftrightarrow \quad (M_0, *, M_0) \in \beta_0^{op}(b) \\ &\qquad \text{by the definition of } an \text{ and } na, \\ &\Leftrightarrow \quad b \in M_0 \text{ by the definition of } \beta_0. \end{aligned}$$

From the equivalence

$$\beta_0^{op}(b) \in M'_0 \Leftrightarrow b \in M_0$$

we deduce $\beta_0 M_0 = M'_0$, that β_0 preserves initial marking. In addition β_0 preserves pre and post conditions of events from II, I of lemma 54. □

Now we establish the adjunction between **A** and **N** in which an is left adjoint to na.

Lemma 61 *Let $T = (S, i, E, I, Tran)$ be an asynchronous transition system and $N = (B, M_0, E', pre, post)$ a net.*

For a morphism of nets $(\beta, \eta) : an(T) \to N$, defining $\sigma(s) = \beta \circ M(s)$, for $s \in S$, yields a morphism of asynchronous transition systems

$$\theta(\beta, \eta) =_{def} (\sigma, \eta) : T \to na(N).$$

For a morphism of asynchronous transition systems $(\sigma, \eta) : T \to na(N)$, defining

$$c\beta b \text{ iff } \emptyset \neq c = \{(s, e, s') \in Tran_* \mid b \in \sigma(s) \,\&b \in \sigma(s') \;\&\; b \notin^\bullet \eta(e)^\bullet\},$$

yields a morphism

$$\varphi(\sigma, \eta) =_{def} (\beta, \eta) : an(T) \to N.$$

Furthermore, θ and φ are mutual inverses, establishing a bijection between morphisms

$$an(T) \to N$$

and

$$T \to na(N).$$

Proof: First note $\theta(\beta, \eta)$ and $\varphi(\sigma, \eta)$ above are morphisms because they are the compositions

$$\theta(\beta, \eta) : \quad T \stackrel{(\sigma_0, 1_E)}{\longrightarrow} na \circ an(T) \stackrel{na(\beta,\eta)}{\longrightarrow} na(N)$$

$$\varphi(\sigma, \eta) : \quad an(T) \stackrel{an(\sigma,\eta)}{\longrightarrow} an \circ na(N) \stackrel{(\beta_0, 1_{E'})}{\longrightarrow} N$$

with the "unit" and "counit" morphisms of lemmas 59, 60. We require that θ, φ form a bijection.

Letting $(\sigma, \eta) : T \to na(N)$, we require $\theta \circ \varphi(\sigma, \eta) = (\sigma, \eta)$. We know $\theta \circ \varphi(\sigma, \eta)$ has the form (σ', η). Writing $(\beta, \eta) =_{def} \varphi(\sigma, \eta)$ we have $\sigma'(s) = \beta \circ M(s)$ for any $s \in S$. Now note

$$\begin{aligned} b' \in \sigma'(s) &\Leftrightarrow b' \in \beta \circ M(s) \\ &\Leftrightarrow \beta^{op}(b') \in M(s) \\ &\Leftrightarrow (s, *, s) \in \beta^{op}(b') \\ &\Leftrightarrow b' \in \sigma(s) \end{aligned}$$

where the final equivalence follows from the definition of φ, recalling $(\beta, \eta) = \varphi(\sigma, \eta)$. Thus $\sigma' = \sigma$ and hence $\theta \circ \varphi(\sigma, \eta) = (\sigma, \eta)$.

To complete the proof, it is necessary to show $\varphi \circ \theta(\beta, \eta) = (\beta, \eta)$ for an arbitrary morphism $(\beta, \eta) : an(T) \to N$. Then, write $(\beta', \eta) =_{def} \varphi \circ \theta(\beta, \eta)$. To show $\beta' = \beta$, consider an arbitrary $(s, e, s') \in Tran_*$. Let $b \in B$. From the definitions of θ and φ,

$$(s, e, s') \in \beta'^{op}(b) \Leftrightarrow b \in \beta M(s) \ \&\ b \in \beta M(s') \ \&\ b \notin^{\bullet} \eta(e)^{\bullet}. \qquad (\dagger)$$

Note that

$$\begin{aligned} b \in \beta M(s) &\Leftrightarrow \beta^{op}(b) \in M(s) \\ &\Leftrightarrow (s, *, s) \in \beta^{op}(b). \end{aligned}$$

Note too that, as (β, η) is a morphism,

$$b \in^{\bullet} \eta(e)^{\bullet} \Leftrightarrow \beta^{op}(b) \in^{\bullet} e^{\bullet}.$$

Hence, rewriting $(\dagger)$,

$$(s, e, s') \in \beta'^{op}(b) \Leftrightarrow (s, *, s) \in \beta^{op}(b) \ \&\ (s', *, s') \in \beta^{op}(b) \ \&\ \beta^{op}(b) \notin^{\bullet} e^{\bullet}.$$

However, under the assumption that $(s, *, s)$ and $(s', *, s')$ belong to $\beta^{op}(b)$ we have

$$\beta^{op}(b) \notin^{\bullet} e^{\bullet} \Leftrightarrow (s, e, s') \in \beta^{op}(b).$$

(Recall the definition of ${}^{\bullet}e$ and $e^{\bullet}$ in $an(T)$.)

Thus

$$(s, e, s') \in \beta'^{op}(b) \Leftrightarrow (s, e, s') \in \beta^{op}(b).$$

Consequently, $\beta' = \beta$, and we conclude $\varphi \circ \theta(\beta, \eta) = (\beta, \eta)$. $\square$

Theorem 62 *The functors $an : \mathbf{A} \to \mathbf{N}$ and $na : \mathbf{N} \to \mathbf{A}$ form an adjunction with an left adjoint to na; the components of the units and counits of the adjunction are the morphisms given in lemmas 59, 60.*

Proof: Let T be an asynchronous transition system and N a net. Let $(\sigma_0, 1_E) : T \to na \circ an(T)$ be the morphism described in lemma 59. Let $(\sigma, \eta) : T \to na(N)$ be a morphism in **A**. Then, because of the bijection, $\varphi(\sigma, \eta)$ is the unique morphism $h : an(T) \to N$ such that

$$(\sigma, \eta) = \theta(h) = na(h) \circ (\sigma_0, 1_E)$$

—as remarked in the proof of lemma 61, $\theta(h)$ is this composition. $\square$

10.2.2 A coreflection

Neither **A** nor **N** embeds fully and faithfully in the other category via the functors of the adjunction. This accompanies the facts that neither unit nor counit is an isomorphism (see [50] p.88); in passing from a net N to $an \circ na(N)$ extra conditions are most often introduced; the net $an \circ na(N)$ is always safe, as we will see. While passing from an asynchronous transition system T to $na \circ an(T)$ can, not only blow-up the number of states, but also collapse states which cannot be separated by conditions.

A (full) coreflection between asynchronous transition systems and nets can be obtained at the cost of adding three axioms. Let $\mathbf{A}_0$ be the full subcategory of asynchronous transition systems $T = (S, i, E, I, Tran)$ satisfying the following:

Axiom 1 Every state is reachable from the initial state, *i.e.* for every $s \in S$ there is a chain of events $e_1, \ldots, e_n$, possibly empty, for which $i \xrightarrow{e_1 \cdots e_n} s$, where i is the initial state.

Axiom 2 $M(u) = M(s) \Rightarrow u = s$, for all $s, u \in S$.

Axiom 3 ${}^\bullet e \subseteq M(s) \Rightarrow \exists s'. \, (s, e, s') \in Tran$, for all $s \in S, e \in E$.

Axioms 2 and 3 enforce two separation properties. The contraposition of Axiom 2 says

$$u \neq s \Rightarrow M(u) \neq M(s)$$

i.e. that if two states are distinct then there is a condition of T holding at one and not the other. In fact, Axiom 2 is equivalent to

$$u \neq s \Rightarrow \exists b. \, b \in M(u) \;\&\; b \notin M(s)$$

though we postpone the justification of this till after we have treated *complementation* of conditions. We can recast Axiom 3 into the following form when it becomes more clearly a separation axiom: If (u, e, u') is a transition and s is a state from which there is no e-transition then there is a condition b of T such that

$$b \in M(u) \;\&\; (u, e, u') \notin b \;\&\; b \notin M(s).$$

Axioms 2 and 3 hold for any asynchronous transition system $na(N)$ got from a net N. The proof that Axiom 3 holds uses the operation of *complementation* on conditions of an asynchronous transition system. The properties of complementation are listed below:

Proposition 63 *Let b be a condition of an asynchronous transition system $T = (S, i, E, I, Tran)$. Define*

$$\overline{b} = \{(s, e, s') \in Tran_* \mid (s, e, s') \notin b \ \&\ (s, *, s) \notin b \ \&\ (s', *, s') \notin b\}.$$

If nonempty, $\overline{b}$ is a condition of T. It has the following properties:

$$\begin{array}{rcl} (s, *, s) \in \overline{b} & \Leftrightarrow & (s, *, s) \notin b, \text{ for any } s \in S, \\ \overline{b} \in {}^{\bullet}e & \Leftrightarrow & b \in e^{\bullet} \ \&\ b \notin {}^{\bullet}e \\ \overline{b} \in e^{\bullet} & \Leftrightarrow & b \in {}^{\bullet}e \ \&\ b \notin e^{\bullet} \text{ for any } e \in E. \end{array}$$

Let $(\sigma, \eta) : T' \to T$ be a morphism of asynchronous transition systems and b be a condition of T. Then

$$(\sigma, \eta)^{-1}\overline{b} = \overline{(\sigma, \eta)^{-1}b}.$$

Suppose u, s are two distinct markings of a net N. Then certainly there is a condition b of the net in one but not the other.

Suppose for instance $b \notin u$ and $b \in s$. Then, from the way the extent of a condition is defined,

$$|b| \notin M(u) \text{ and } |b| \in M(s).$$

With complementation we can separate the other way:

$$\overline{|b|} \in M(u) \text{ and } \overline{|b|} \notin M(s).$$

This justifies our earlier remark that that Axiom 2 is equivalent to the seemingly stronger axiom:

$$u \neq s \Rightarrow \exists b.\ b \in M(u) \ \&\ b \notin M(s)$$

We return to the verification that the asynchronous transition system $na(N)$ of a net N satisfies Axioms 2 and 3.

Proposition 64 *Let $N = (B, M_0, E, pre, post)$ be a net. Then $na(N)$ satisfies the Axioms 2 and 3 above.*

Proof: If u, s are distinct states of $na(N)$ they are distinct markings of N and hence only one contains some condition b. But then $|b|$ can only be an element of one of $M(u)$ and $M(s)$ which are therefore unequal. This demonstrates (the contraposition of) Axiom 2.

Now we show $na(N)$ satisfies the contraposition of Axiom 3. Supposing $u \xrightarrow{e} u'$ and $s \overset{e}{\not\to}$ in N, we are required to exhibit a condition c of $na(N)$ such that

$$c \in {}^{\bullet}e \ \ \&\ \ c \notin M(s).$$

There are two ways in which the marking s can fail to enable event e. Either

(i) $pre(e) \not\subseteq s$ or

(ii) $post(e) \cap (s \setminus pre(e)) \neq \emptyset$.

In the case of (i), there is a condition $b \in B$ of the net such that

$$b \in pre(e) \ \& \ b \notin s.$$

Hence

$$|b| \in {}^{\bullet}e \ \& \ |b| \notin M(s).$$

In the case of (ii), there is a condition $b \in B$ of the net such that

$$b \in post(e) \ \& \ b \in s \ \& \ b \notin pre(e)$$

Hence

$$|b| \in e^{\bullet} \ \& \ |b| \in M(s) \ \& \ |b| \notin {}^{\bullet}e.$$

But then, taking the complement of $|b|$,

$$\overline{|b|} \in {}^{\bullet}e \ \& \ \overline{|b|} \notin M(s),$$

by proposition 63.

In either case, (i) or (ii), we obtain a condition c of $na(N)$ for which

$$c \in {}^{\bullet}e \ \& \ c \notin M(s).$$

$\square$

Recall a net is *safe* if for each reachable marking M and event e

$${}^{\bullet}e \subseteq M \Rightarrow e^{\bullet} \cap (M \setminus {}^{\bullet}e) = \emptyset.$$

As we now see, if T is an asynchronous transition system which satisfies Axioms 2 and 3 then $an(T)$ is a safe net whose behaviour is seen to be isomorphic to that of T on reachable states.

Lemma 65 *Assume $T = (S, i, E, I, Tran)$ is an asynchronous transition system satisfying Axioms 2 and 3 above. Then*

1. $e_1 I e_2 \Leftrightarrow {}^{\bullet}e_1{}^{\bullet} \cap {}^{\bullet}e_2{}^{\bullet} = \emptyset$ *in* $an(T)$, *for any events* e_1, e_2,

2. $(s, e, s') \in Tran \Leftrightarrow M(s) \xrightarrow{e} M(s')$ in $an(T)$ *for any* $s, s' \in S$ *and* $e \in E$,

3. *$an(T)$ is a safe net in which every reachable marking has the form $M(s)$ for some state s of T.*

Proof: By lemma 57,

$$e_1 I e_2 \Leftrightarrow {}^\bullet e_1{}^\bullet \cap {}^\bullet e_2{}^\bullet = \emptyset,$$
$$(s,e,s') \in \mathit{Tran} \Rightarrow M(s) \xrightarrow{e} M(s') \text{ in } an(T).$$

This yields (1) and (2)"$\Rightarrow$". To establish the converse, (2)"$\Leftarrow$", with the assumption of Axioms 2 and 3, suppose $M(s) \xrightarrow{e} M(s')$. Then ${}^\bullet e \subseteq M(s)$ so $(s,e,s_1) \in \mathit{Tran}$ from some state s_1 by Axiom 3. Thus $M(s) \xrightarrow{e} M(s_1)$ and so $M(s') = M(s_1)$. Now by Axiom 2 we deduce $s' = s_1$, and hence the converse

$$M(s) \xrightarrow{e} M(s') \Rightarrow (s,e,s') \in \mathit{Tran}.$$

We now show (3). Any reachable marking of $an(T)$ has the form $M(s)$ for some $s \in S$ by the following argument: Assuming $M(s) \xrightarrow{\;e\;} M_1$ we necessarily have ${}^\bullet e \subseteq M(s)$ whereupon, as above, there is a transition (s,e,s_1) of T with $M_1 = M(s_1)$; thus, by induction along any reachability chain, any reachable marking of $an(T)$ is of the form $M(s)$ for some state s of T. Because the two sets

$$\begin{aligned} e^\bullet &= \{b \in M(s') \mid (s,e,s') \notin b\}, \\ M(s) \setminus {}^\bullet e &= \{b \in M(s) \mid (s,e,s') \in b\} \end{aligned}$$

are clearly disjoint, the net $an(T)$ is safe. □

Corollary 66 *For any net N, the net $an \circ na(N)$ is safe.*

The coreflection between $\mathbf{A}_0$ and $\mathbf{N}$ is defined using a simple coreflection between the full subcategory of $\mathbf{A}$, consisting of objects, where all states are reachable, and $\mathbf{A}$.

Definition: Let $\mathbf{A}^R$ be the full subcategory of $\mathbf{A}$ consisting of asynchronous transition systems (S,i,E,I,Tran) satisfying Axiom 1, *i.e.* so that all states s are reachable.

Let $\mathcal{R}$ act on an asynchronous transition system $T = (S,i,E,I,\mathit{Tran})$ as follows:

$$\mathcal{R}(T) = (S', i', E', I', \mathit{Tran}')$$

where

$$\begin{aligned} S' & \quad \text{consists of all reachable states of } T \\ E' &= \{e \in E \mid \exists s,s' \in S'.\, (s,e,s') \in \mathit{Tran}\} \\ I' &= I \cap (E' \times E') \\ \mathit{Tran}' &= \mathit{Tran} \cap (S' \times E' \times S'). \end{aligned}$$

For a morphism $(\sigma,\eta) : T \to T'$ of asynchronous transition systems, define $\mathcal{R}(\sigma,\eta) = (\sigma',\eta')$ where σ' and η' are the restrictions of σ and η to the states, respectively events, of $\mathcal{R}(T)$.

We note that a morphism from an asynchronous transition system in which all states are reachable is determined by how it acts on events:

Proposition 67 *Suppose (σ, η) and (σ', η) are morphisms $T \to T'$ between asynchronous systems where each state of T is reachable. Then $\sigma = \sigma'$.*

Proof: An obvious consequence of the determinacy property

$$(s, e, s_1) \in \mathit{Tran} \ \& \ (s, e, s_2) \in \mathit{Tran} \Rightarrow s_1 = s_2$$

of asynchronous transition systems. $\square$

Proposition 68 *The operation $\mathcal{R}$ is a functor $\mathbf{A} \to \mathbf{A}^R$ which is right adjoint to the inclusion functor $\mathcal{I} : \mathbf{A}^R \to \mathbf{A}$. The unit of the adjunction at $T \in \mathbf{A}^R$ is the identity on T, making the adjunction a coreflection. The counit at $T \in \mathbf{A}^R$ is given by $(j_S, j_E) : R(T) \to T$ where j_S and j_E are the inclusion maps on states and events respectively. Moreover, $\mathcal{R}$ preserves Axioms 2 and 3 in the sense that if T satisfies Axiom 2 (or 3) then $\mathcal{R}(T)$ satisfies Axiom 2 (or 3).*

Proof: We omit the straightforward proof that $\mathcal{R}$ is a right adjoint to the inclusion of categories with counit as claimed. Let $j : \mathcal{R}(T) \to T$ be a component of the counit. The transitions Tran' of $\mathcal{R}(T)$ are a subset of those of T. If b is a condition of T then $j^{-1}b = b \cap \mathit{Tran}'$ is a condition of $\mathcal{R}(T)$ provided it is nonempty. Suppose s_1 and s_2 are two distinct states of $\mathcal{R}(T)$. If T satisfies Axiom 2 then there is a condition b of T such that one and only one of $(s_1, *, s_1) \in b$, $(s_2, *, s_2) \in b$ holds. But then $j^{-1}b$ is a condition of $\mathcal{R}(T)$ separating s_1, s_2. Thus $\mathcal{R}$ preserves Axiom 2, and by a similar argument, Axiom 3. $\square$

We show the adjunction, with *an* left adjoint to $\mathcal{R} \circ na$, obtained as the composition forms a coreflection. Its counit is given by the notion of *reachable extent* of a condition. This consists essentially of the reachable markings and transitions at which b holds uninterruptedly.

Definition: Let N be a net. Let Tran_* be the transitions and idle transitions of $\mathcal{R} \circ na(N)$. Define

$$|b|^R = |b| \cap \mathit{Tran}_*.$$

Theorem 69 *Defining $na_0 = \mathcal{R} \circ na$, the composition of functors, yields a functor $na_0 : \mathbf{N} \to \mathbf{A}_0$ which is right adjoint to $an_0 : \mathbf{A}_0 \to \mathbf{N}$, the restriction of an to $\mathbf{A}_0$.*

The unit at $T = (S, i, E, I, \mathit{Tran}) \in \mathbf{A}_0$ is an isomorphism

$$(\sigma, 1_E) : T \to na_0 \circ an(T)$$

where $\sigma(s) = M(s)$ for $s \in S$, making the adjunction a coreflection.

The counit at a net N is

$$(\beta, 1_E) : an \circ na_0 \to N$$

where

$$c\beta b \textit{ iff } \emptyset \neq c = |b|^R$$

between conditions c of $na_0(N)$ and b of N.

Proof: The adjunctions compose to give $\mathcal{R} \circ na : \mathbf{N} \to \mathbf{A}^R$ a right adjoint to $\mathcal{I} \circ an : \mathbf{A}^R \to \mathbf{N}$. However, the image $\mathcal{R} \circ na(N)$ of a net N always satisfies Axioms 2 and 3 as well as 1. This is because $na(N)$ satisfies Axioms 2 and 3, and $\mathcal{R}$ preserves these axioms. Thus the adjunction cuts down to one where $na_0 : \mathbf{N} \to \mathbf{A}_0$ is right adjoint to $an_0 : \mathbf{A}_0 \to \mathbf{N}$. It is an adjunction with unit at $T = (S, i, E, I, \mathit{Tran}) \in \mathbf{A}_0$ a morphism in $\mathbf{A}_0$

$$(\sigma, 1_E) : T \to na_0 \circ an(T)$$

where $\sigma(s) = M(s)$ for $s \in S$.

That the unit $(\sigma, 1_E) : T \to na_0 \circ an(T)$ is an isomorphism follows from lemma 65. Hence the functors an, na_0 form a coreflection with an_0 left adjoint to na_0.

That the counit has the form claimed follows by composing the natural bijections of the adjunctions given by proposition 68 and lemma 61. □

One consequence of the coreflection is that any net N can be converted to a safe net $an \circ na_0(N)$ with the same behaviour, in the sense that that there is as an isomorphism between reachable asynchronous transition systems the two nets induce under na_0. Another is that $\mathbf{A}_0$ has products and coproducts given by the same constructions as those of $\mathbf{A}$.

Proposition 70 *The category $\mathbf{A}_0$ has products and coproducts which coincide with those in the category $\mathbf{A}$.*

Proof: The product of nets in $\mathbf{N}$ becomes the product in $\mathbf{A}_0$ of asynchronous transition systems under na_0. Its behaviour, which is described in proposition 48, ensures that its image under na_0 coincides with the product in $\mathbf{A}$.

The coproduct in $\mathbf{A}$ will be the coproduct in $\mathbf{A}_0$ provided it is the image to within isomorphism of a net. However, if T_0, T_1 are objects of $\mathbf{A}_0$, then by lemma 46 their coproduct in $\mathbf{A}$ is isomorphic to $na_0(an(T_0) + an(T_1))$. □

The coreflection $\mathbf{A}_0 \hookrightarrow \mathbf{N}$ cuts down to an equivalence of categories by restricting to the appropriate full subcategory of nets.

Definition: Let $\mathbf{N}_0$ be the full subcategory on nets such that

$$b \mapsto |b|^R$$

is a bijection between conditions of N and those of $na_0(N)$.

Theorem 71 *The functor an restricts to a functor $an_0 : \mathbf{A}_0 \to \mathbf{N}_0$. The functor $\mathcal{R} \circ na$ restricts to a functor $na_0 : \mathbf{N}_0 \to \mathbf{A}_0$. The functors an_0, na_0 form an equivalence of categories.*

Proof: Recall the coreflection of theorem 69: $na_0 = \mathcal{R} \circ na : \mathbf{N} \to \mathbf{A}_0$ is right adjoint to $an_0 : \mathbf{A}_0 \to \mathbf{N}$, the restriction of an to $\mathbf{A}_0$. The counit of the coreflection, at a net N,

$$(\beta, 1_E) : an_0 \circ na_0(N) \to N$$

has $c\beta b$ iff $c = |b|^R$, between condition. This is an isomorphism iff $N \in \mathbf{N}_0$. We thus obtain an equivalence of categories. □

Nets in $\mathbf{N}_0$ are saturated with conditions in the sense that they have as many conditions as is allowed by their reachable behaviour and independence (regarded as an asynchronous transition system). Nets in $\mathbf{N}_0$ cannot however have more than one copy of a condition with particular starting and ending events (they are *condition-extensional*). This is because:

Proposition 72 *Let T be an asynchronous transition system for which each state is reachable. If b_1, b_2 are conditions of T for which*

$$^\bullet b_1 = ^\bullet b_2 \quad \text{and} \quad b_1^\bullet = b_2^\bullet$$

then

$$b_1 = b_2.$$

Proof: Suppose $^\bullet b_1 = ^\bullet b_2$ and $b_1^\bullet = b_2^\bullet$ for conditions b_1, b_2 of T. Inductively along a chain of transitions

$$(i, e_1, s_1), (s_1, e_2, s_2), \ldots, (s_{n-1}, e_n, s_n)$$

the membership of (s_{i-1}, e_i, s_i) (or $(s_i, *, s_i)$) in b_1 and in b_2 must agree. □

If on the other hand an asynchronous transition system T has a state which is not reachable then there will be distinct conditions of T with the same end points. Suppose T has states which are not reachable let $Tran_0$ be all transitions, including idle ones, which are not reachable. If b_1 is a condition, say consisting solely of reachable transitions of T, then so is $b_2 = b_1 \cup Tran_0$ a condition, necessarily distinct from b_1, but with $^\bullet b_1 = ^\bullet b_2$ and $b_1^\bullet = b_2^\bullet$.

We have already observed the coreflection from event structures to asynchronous transition systems $\mathbf{E} \hookrightarrow \mathbf{A}$. In fact the coreflection cuts down to one between $\mathbf{E} \hookrightarrow \mathbf{A}_0$.

Proposition 73 *For any event structure E, the asynchronous transition system $tla \circ etl(E)$ is an object in $\mathbf{A}_0$. Consequently, $tla \circ etl$ cuts down to a functor $\mathbf{E} \to \mathbf{A}_0$ left adjoint to the restriction of $atl \circ tle$ to $\mathbf{A}_0 \to \mathbf{E}$ also forming a coreflection.*

Proof: The functor $tla \circ etl : \mathbf{E} \to \mathbf{A}$ is left adjoint to $atl \circ tle : \mathbf{A} \to \mathbf{E}$ and forms a coreflection. It suffices to show that $tla \circ etl(E)$ is an object of $\mathbf{A}$ for any event structure E. Let E be a event structure. Note that $tla \circ etl(E)$ is an asynchronous transition system isomorphic to the asynchronous transition system with transitions $x \xrightarrow{e} x'$, between finite configurations of E, and independence relation *co*—see proposition 52. There are many ways of adjoining conditions to events of an event structure so as to produce a (safe) net N with reachable transition and independence relations isomorphic to that of the configurations of E (see *e.g.* the construction of an occurrence net from an event structure in [64]). Hence by theorem 69, $na_0(N)$, and so the isomorphic $tla \circ etl(E)$, belong as objects to $\mathbf{A}_0$. $\square$

10.3 Properties of conditions

In this section we explore briefly the "logical" properties of conditions of an asynchronous transition system. These follow from the construction of conditions out of special subsets. A general condition can be decomposed into a disjoint union of minimal components, called connected conditions. As will be seen, the behaviour of the net constructed from an asynchronous transition system is determined by just its connected conditions. The notion of connected condition appears automatically in establishing an adjunction between asynchronous transition systems and a previously studied category of safe nets. This category of nets has a broader class of morphisms, ones which allow general foldings.

Let us first summarise the properties of conditions of an asynchronous transition system that have arisen in the proofs above:

- If b is a condition of an asynchronous transition system T and $(\sigma, \eta) : T' \to T$ is a morphism of asynchronous transition systems, then the inverse image $(\sigma, \eta)^{-1} b$ is a condition of T', if nonempty (*cf.* lemma 55).

- If X is a nonempty pairwise dependent subset of events of an asynchronous transition system T (so $e_1 I e_2$ for all $e_1, e_2 \in X$) then there is a condition of T started by precisely the events X and ended by precisely X (*cf.* lemma 56).

- Any condition b of an asynchronous transition system has a complement $\overline{b}$, either empty or a condition whose starting and ending events reverse those of b (*cf.* proposition 63).

In general, conditions of an asynchronous transition system are not closed under intersections and unions. However in the case where the unions are disjoint:

- If S be a nonempty collection of conditions of an asynchronous transition system T which is disjoint, in the sense that

$$\text{if } b_1 \cap b_2 \neq \emptyset \text{ then } b_1 = b_2, \text{ for all } b_1, b_2 \in S,$$

then $\bigcup S$ is a condition of T.

10.3.1 Connected conditions

A new condition can be built up out of disjoint components. And in fact, conversely, any condition can be decomposed into a disjoint union of *connected* components which cannot be decomposed further.

Definition: Let T be an asynchronous transition system with conditions B. For $b, b' \in B \cup \{\emptyset\}$, write

$$b' \text{ comp } b \Leftrightarrow_{def} b' \subseteq b \quad \& \quad b \backslash b' \in B \cup \{\emptyset\}.$$

(If b' comp b we say b' is a *component* of b.)

Say a condition b is *connected* iff there are *not* conditions b_1, b_2 such that

$$b = b_1 \cup b_2 \quad \& \quad b_1 \cap b_2 = \emptyset.$$

Notation: We shall write $b = b_1 \dot{\cup} b_2$ to mean $b = b_1 \cup b_2$ and $b_1 \cap b_2 = \emptyset$, for sets b, b_1, b_2.

In fact, a component of a condition b of an asynchronous transition system is either empty or a condition included in b whose boundary of starting/ending events agrees with that of b:

Lemma 74 *Let c, b be conditions of an asynchronous transition system T. Then,*

$$\begin{aligned} c \text{ comp } b \Leftrightarrow c \subseteq b \;\&\; & \textit{for all transitions } (s,e,s') \textit{ of } T \\ & (s,e,s') \in {}^\bullet c \Rightarrow (s,e,s') \in {}^\bullet b \;\& \\ & (s,e,s') \in c^\bullet \Rightarrow (s,e,s') \in b^\bullet. \end{aligned}$$

Proof: By basic set-theory from the definition of a condition of an asynchronous transition system. □

Proposition 75 *Let T be an asynchronous transition system with conditions B.*

(i) The relation comp *is a partial order on $B \cup \{\emptyset\}$.*

(ii) A condition is connected iff it is a minimal condition with respect to comp.

(iii) If c_i, $i \in I$, is a family of components of a condition b, then

$$\bigcap_{i\in I} c_i, \quad \bigcup_{i\in I} c_i$$

are also components of b.

(iv) If c_1 and c_2 are components of a condition b, then $c_1 \backslash c_2$ is also a component of b.

Proof:
(i) Clearly comp is reflexive. Suppose c comp b comp a. Then $c \subseteq b$ and $b \setminus c$ is a condition if nonempty as well as $b \subseteq a$ and $a \setminus b$ forming a condition if it is nonempty. It follows that $c \subseteq a$ and that

$$a \setminus c = (a \setminus b) \mathbin{\dot\cup} (b \setminus c)$$

and hence $a \setminus c$ is itself a condition if nonempty. But this makes c comp a.
(ii) Clear.
(iii) Assume $c_i, i \in I$ is a family of components of a condition b. It can be checked that $\bigcup_{i\in I} c_i$ and $\bigcap_{i\in I} c_i$ satisfy the axioms required of conditions of T, if nonempty.

To give the flavour of the arguments, assume $(s, e, s') \in (\bigcup_i c_i)^\bullet$ and that (u, e, u') is a transition of which we wish to show $(u, e, u') \in (\bigcup_i c_i)^\bullet$—one of the properties needed for the union to be a condition. As $(s, e, s') \in (\bigcup_i c_i)^\bullet$, there is a component c_j for which $(s, e, s') \in {c_j}^\bullet$. But then $(s, e, s') \in b^\bullet$ (lemma 74). As b is a condition, $(u, e, u') \in b^\bullet$ and, in particular, $(u, e, u') \notin b$. As c_j is a condition, $(u, e, u') \in {c_j}^\bullet$, and in particular $(u, *, u) \in c_j$. But then $(u, *, u) \in \bigcup_i c_i$, and $(u, e, u') \notin \bigcup_i c_i$ as $\bigcup_i c_i \subseteq b$. This makes $(u, e, u') \in (\bigcup_i c_i)^\bullet$.

The conditions $\bigcup_{i\in I} c_i$ and $\bigcap_{i\in I} c_i$ are also components because the complements,

$$b \setminus \bigcup_i c_i = \bigcap_i (b \setminus c_i)$$
$$b \setminus \bigcap_i c_i = \bigcup_i (b \setminus c_i)$$

and, as we have just remarked, intersections and unions of components of b form conditions if nonempty.
(iv) Finally, suppose c_1 and c_2 are components of b. Then $(b \setminus c_1) \cup c_2$ is a union of components of b, and hence itself a component, ensuring $b \setminus ((b \setminus c_1) \cup c_2)$ is a component of b. Note $c_1 \setminus c_2 = b \setminus ((b \setminus c_1) \cup c_2)$. □

Definition: Let b be a condition of an asynchronous transition system. Define

$$\mathrm{conn}(b) = \{c \mid c \text{ is a connected condition } \& \ c \text{ comp } b\}.$$

The family of connected components conn(b), of a condition b is disjoint and covers b in the following sense:

Lemma 76 *Given a condition b of an asynchronous transition system,*

(i) $c_1 \in \mathrm{conn}(b)$ & $c_2 \in \mathrm{conn}(b)$ & $c_1 \cap c_2 \neq \emptyset \Rightarrow c_1 = c_2$, *and*

(ii) $b = \bigcup \mathrm{conn}(b)$.

Proof:
(i) Suppose $c_1, c_2 \in \mathrm{conn}(b)$ and $c_1 \cap c_2 \neq \emptyset$. It follows that $c_1 \cap c_2$ and $c_1 \setminus c_2$ are components of b if nonempty, with

$$c_1 = (c_1 \setminus c_2) \,\dot{\cup}\, (c_1 \cap c_2).$$

But c_1 is connected so $c_1 \setminus c_2 = \emptyset$ yielding $c_1 \subseteq c_2$. Similarly, $c_2 \subseteq c_1$, implying $c_1 = c_2$.
(ii) Clearly $\bigcup \mathrm{conn}(b) \subseteq b$. To show the converse inclusion, let $t \in b$. Take

$$d = \bigcap \{c \mid t \in c \;\&\; c \text{ comp } b\}.$$

Being an intersection of b's components, d is itself a component of b. Clearly it contains t, is a minimal component of b, and so is connected. Thus $t \in d$ and $d \in \mathrm{conn}(b)$, ensuring $t \in \bigcup \mathrm{conn}(b)$. $\square$

Recall the construction $an(T)$ of a net from an asynchronous transition system T. When T is an object of $\mathbf{A}_0$, the net $an(T)$ is safe and has behaviour isomorphic to that of T (*cf.* lemma 65). In fact we can restrict the construction to just the connected conditions—these are sufficient to determine the net's behaviour.

Definition: Let T be an asynchronous transition system in $\mathbf{A}_0$. Assume $an(T)$ has the form $(B, M_0, E, pre, post)$. Define $an_c(T) = (C, M_0 \cap C, E, pre_c, post_c)$ where C consists of the connected conditions of T and

$$pre_c(e) = pre(e) \cap C, \quad post_c(e) = post(e) \cap C.$$

Lemma 77 *Let T be an asynchronous transition system in $\mathbf{A}_0$. Then:*

(i) $an_c(T)$ is a safe net.

(ii) There is an isomorphism $(\sigma, 1_E) : T \to na_0 \circ an_c(T)$ where $\sigma(s) = M(s) \cap C$ for s a state of T with connected conditions C.

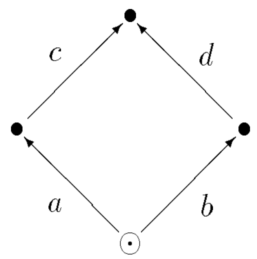
Proof: The separation axioms Axiom 2 and Axiom 3 hold iff they hold restricted to only connected conditions. For this reason the proof proceeds pretty much as that of lemma 65. There are a couple of refinements. First showing that $an_c(T)$ is a net requires that $pre_c(e)$ and $post_c(e)$ are nonempty for any event e. As earlier, in the proof of lemma 57, lemma 56 yields b a pre and post-condition of

e, though it is not necessarily connected; however amongst its connected components conn(b) are connected pre and post-conditions of e. An extra argument is needed in showing that independence of the net $an_c(T)$ coincides with that of T, as is required for the isomorphism of (ii). This involves showing that if $\neg e_1 I e_2$ in T then there is a *connected* condition c in ${}^\bullet e_1{}^\bullet \cap {}^\bullet e_2{}^\bullet$. By lemma 56, there is a condition b (not necessarily connected) such that

$$\{e_1, e_2\} = \{e \mid b \in {}^\bullet e\} = \{e \mid b \in e^\bullet\}.$$

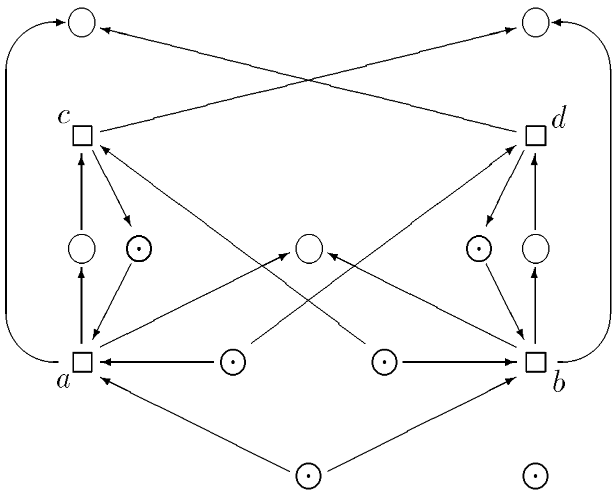
As T is a asynchronous transition system in which all transitions are reachable, it has transitions (s_1, e_1, s_1') and (s_2, e_2, s_2') that are connected by a chain of transitions (backwards and then forwards from the initial state) which do *not* involve the events e_1, e_2. This chain must be included in a single connected component c of b—otherwise, by lemma 74, c and so b would be started or ended by an event other than e_1 or e_2. This connected component is one for which $c \in {}^\bullet e_1{}^\bullet \cap {}^\bullet e_2{}^\bullet$. $\square$

Example: The asynchronous transition system



cId

gives rise, under an_c, to the following net, in which every condition is connected:



10.3.2 Relational morphisms on nets

With the aid of connected conditions we can link to another definition of morphism on nets. A limitation with the definition of morphism on nets (in section 9.1) is that it does not permit "folding" morphisms of the kind illustrated in the example below.

Let us temporarily broaden our attention to Petri nets in which conditions can hold with multiplicities, in other words to general Petri nets, though without capacities. Typically such a net consists of

$$(B, M_0, E, pre, post)$$

with much the same intuition as before, but where the initial marking M_0 is now a *multiset* of the set of conditions B and *pre* and *post* are now *multirelations* specifying the multiplicity of conditions used or produced by an event in E. Letting M, M' be markings (i.e. multisets of conditions) and A be a finite multiset of events we define $M \xrightarrow{A} M'$ iff

$$pre.A \leq M \text{ and } M' = M - (pre.A) + (post.A).$$

Here we are using a little multiset notation. Thinking of multisets as vectors (of possibly infinite dimension) and multirelations as matrices (possibly infinite), we can compare two multisets coordinatewise with the relation $\leq$, add and subtract multisets (provided no component goes negative). Using matrix multiplication we can apply a multirelation to a multiset as in $pre.A$ which yields the multiset of preconditions of the multiset of events A. The fact that A is a multiset reveals concurrency amongst event occurrences; for example a transition

$$M \xrightarrow{3e+2f} M'$$

is interpreted as meaning that three occurrences of the event e occur concurrently with each other and with two of f in taking the marking M to M'.[9]

What about morphisms between two general Petri nets $N = (B, M_0, E, pre, post)$, $N' = (B', M_0', E', pre', post')$? Taking account of multiplicities, and at the same time respecting events, it is reasonable to take a morphism to be $(\beta, \eta) : N \to N'$ where β is a multirelation from B to B' and η is a partial function from E to E' (which in this context will be understood as a special kind of multirelation) such that

$$\begin{aligned} \beta.M_0 &= M_0' \\ \beta(pre.e) &= pre'.(\eta.e) \text{ and} \\ \beta(post.e) &= post'.(\eta.e), \text{ for all } e \in E. \end{aligned}$$

Such morphisms preserve behaviour:

[9] A thorough treatment of multisets and multirelations can be found in the appendix of [96].

Proposition 78 *Let* $(\beta, \eta) : N \to N'$ *be a morphism of general nets. If* M *is a reachable marking of* N *and* $M \stackrel{A}{\to} M'$ *in* N*, then* $\beta.M$ *is a reachable marking of* N' *and* $\beta.M \stackrel{\eta.A}{\to} \beta.M'$ *in* N'*.*

Proof: A straightforward induction on the number of steps to the reachable marking M. $\square$

Provided we restrict attention to general Petri nets in which every condition occurs with nonzero multiplicity either in the initial marking or the pre- or postconditions of some event, we can form a category by taking the composition of

$$(\beta, \eta) : N \to N' \text{ and } (\beta', \eta') : N' \to N''$$

to be $(\beta' \circ \beta, \eta' \circ \eta)$ where $\beta' \circ \beta$ is the multirelation composition of β' and β, and $\eta' \circ \eta$ is a composition of partial functions—without the restriction the composition could yield infinite multiplicities.

This whole set-up makes sense for safe nets, and we can define **Safe** to be the full subcategory of general Petri nets in which objects are safe nets for which every condition belongs either to the initial marking or the pre- or postcondition of some event. For such safe nets if M is a reachable marking and $M \stackrel{A}{\to} M'$ then the multiset of events A has no multiplicity exceeding 1, *i.e.* A can be identified with a subset of events, which are to be thought of as occurring concurrently. For later, we define the concurrency notation on events e_1, e_2 by taking

$$e_1 \ co\ e_2 \Leftrightarrow_{def} M \xrightarrow{\{e_1, e_2\}} M' \text{ for some reachable markings } M, M'.$$

In a safe net the relation $e_1 \ co\ e_2$ amounts to there existing a reachable marking M for which

$$^{\bullet}e_1 \subseteq M \quad \& \quad ^{\bullet}e_2 \subseteq M \quad \& \quad ^{\bullet}e_1 \cap {}^{\bullet}e_2 = \emptyset,$$

or equivalently, the two events e_1, e_2 are independent and both have concession at some reachable marking.

A little work shows that morphisms in **Safe**, between $N = (B, M_0, E, pre, post)$, $N' = (B', M_0', E', pre', post')$, can be given equivalently as $(\beta, \eta) : N \to N'$ where $\beta \subseteq B \times B'$ is a relation and $\eta : E \to_* E'$ such that

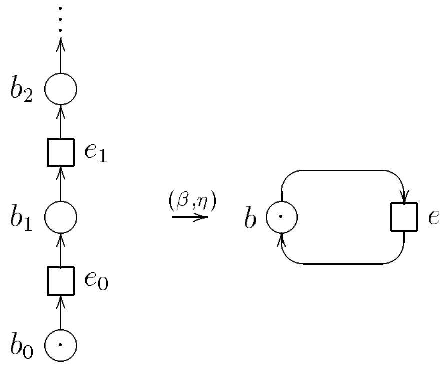
$$\beta M_0 \subseteq M_0' \text{ and } \forall b' \in M_0' \ \exists! b \in M_0.\ b \beta b',$$

and for any event $e \in E$,

$$\beta pre(e) \subseteq pre'(\eta(e)) \text{ and } \forall b' \in pre'(\eta(e)) \ \exists! b \in pre(e).\ b\beta b', \text{ and}$$
$$\beta post(e) \subseteq post'(\eta(e)) \text{ and } \forall b' \in post'(\eta(e)) \ \exists! b \in post(e).\ b\beta b',$$

where the character of multiset application reappears in the form of uniqueness restrictions local to the initial marking and neighbourhoods of events. These say that β^{op} is a function local to M_0', $pre'(\eta(e))$ and $post'(\eta(e))$. It is not required that β^{op} be a partial function globally. The added generality permits the following kind of morphism:

Example: Consider the "folding" morphism in **Safe**



sending each event $e_0, e_1, \ldots$ to the common event e, and each condition $b_0, b_1, \ldots$ to the condition b, so

$$\eta(e_i) = e \text{ and } b_i \beta b$$

for $i \in \omega$. Note that while this morphism does preserve the concurrency relation *co* it does not preserve independence and, for example we have ${}^\bullet e_1^\bullet \cap {}^\bullet e_3^\bullet = \emptyset$, making the events e_1 and e_2 independent whereas their common image e cannot be independent with itself.

So morphisms in **Safe** do not preserve the independence relation on nets if we interpret independence as disjointness of pre and postconditions. But the morphisms do preserve the concurrency relation. We can obtain a functor from **Safe** to asynchronous transition systems by instead interpreting independence as the concurrency relation on safe nets. The functor will map into the category $\mathbf{A}_c$—the full subcategory of $\mathbf{A}_0$ consisting of objects for which the following property holds of the independence relation I:

$$e_1 I e_2 \Rightarrow s \xrightarrow{e_1} s_1 \quad \& \quad s \xrightarrow{e_2} s_2 \text{ for some states } s, s_1, s_2.$$

Proposition 79 $\mathbf{A}_c$ *is a coreflective subcategory of* $\mathbf{A}_0$.

Proof: The inclusion functor has a right adjoint V which from an object $T = (S, i, E, I, \mathit{Tran})$ in $\mathbf{A}_0$ produces $V(T) = (S, i, E, I', \mathit{Tran})$ in which the independence relation is restricted so

$$e_1 I' e_2 \Leftrightarrow_{def} e_1 I e_2 \;\&\; s \xrightarrow{e_1} s_1 \;\&\; s \xrightarrow{e_2} s_2 \text{ for some states } s, s_1, s_2.$$

$\square$

We use V, the right adjoint of the coreflection $\mathbf{A}_c \hookrightarrow \mathbf{A}_0$, to obtain a functor from **Safe** to $\mathbf{A}_c$. For N in **Safe**, define

$$na_c(N) = V \circ na_0(N),$$

an asynchronous transition system whose states are the reachable markings of N and with independence the concurrency relation on N. For a morphism, $(\beta, \eta) : N \to N'$ in **Safe**, define $na_c(\beta, \eta) = (\sigma, \eta)$ where $\sigma(M) = \beta M$ for any reachable marking M of N.

Via the next lemma, the coreflection between $\mathbf{A}_0 \hookrightarrow \mathbf{N}$ yields a coreflection $\mathbf{A}_c \hookrightarrow$ **Safe**.

Lemma 80 *Let T be an asynchronous transition system in $\mathbf{A}_0$ with events E.*

(i) There is a morphism in **Safe**

$$(\gamma, 1_E) : an_c(T) \to an_0(T)$$

with $c\gamma b \Leftrightarrow c \in \mathrm{conn}(b)$.

(ii) For any morphism $(\beta, \eta) : an_c(T) \to N$ in **Safe***, there is a unique morphism $(\varphi, \eta) : an_0(T) \to N$ in* **N** *such that the following diagram commutes:*

$$\begin{tikzcd} an_c(T) \arrow[r, "(\gamma, 1_E)"] \arrow[dr, "(\beta,\eta)"'] & an_0(T) \arrow[d, "(\varphi,\eta)"] \\ & N \end{tikzcd}$$

Proof: Assume T is an asynchronous transition system and that

$$\begin{aligned} an_0(T) &= (B, M_0, E, pre, post), \\ an_c(T) &= (C, M_0 \cap C, E, pre_c, post_c) \end{aligned}$$

where C consists of the connected conditions of T, and

$$pre_c(e) = pre(r) \cap C, \quad post_c(e) = post(e) \cap C$$

for any event e. For a state s of T, we write $M_c(s) = M(s) \cap C$. In particular, $M_0 \cap C = M_c(i)$, where i is the initial state of T.

(i) We show $(\gamma, 1_E)$ is a morphism in **Safe**, according to definition of this section.

Let $c \in M_c(i)$, where i is the initial state of T. If $c\gamma b$ then $(i, *, i) \in c$, so $(i, *, i) \in b$ giving $b \in M(i)$. Thus $\gamma M_0 \cap C \subseteq M_0$. Suppose $c_1, c_2 \in M_c(i)$ with $c_1\gamma b$ and $c_2\gamma b$. Then $(i, *, i) \in c_1 \cap c_2$ where $c_1, c_2 \in \mathrm{conn}(b)$. By lemma 76, $c_1 = c_2$.

Suppose $c \in pre_c(e)$, for $e \in E$, and $c\gamma b$. Then there is a transition (s, e, s') such that $(s, e, s') \in c^\bullet$. By lemma 74, $(s, e, s') \in b^\bullet$, and hence $b \in pre(e)$. Thus $\gamma pre_c(e) \subseteq pre(e)$. Suppose $c_1, c_2 \in pre_c(e)$ with $c_1\gamma b$ and $c_2\gamma b$. Considering an arbitrary transition (s, e, s'), we must have $(s, *, s) \in c_1 \cap c_2$. As $c_1, c_2 \in \mathrm{conn}(b)$, by lemma 76, $c_1 = c_2$.

A similar argument holds for postconditions, and $(\gamma, 1_E)$ fulfils the requirements of a relational morphism.

(ii) The proof relies on a simple fact about relational morphisms, which is a direct consequence of proposition 78:

> Let $(\beta, \eta) : N \to N'$ be a morphism in **Safe**. If M is a reachable marking of N, then βM is a reachable marking of N' such that $b_1 \beta b'$ and $b_2 \beta b'$ implies $b_1 = b_2$ for all conditions $b_1, b_2 \in M$ and b' of N'.

We return to the proof of (ii). Let $(\beta, \eta) : an_c(T) \to N$ be a morphism in **Safe**. For p a condition of N, we claim that

$$\{c \mid c\beta p\}$$

is a disjoint family of connected conditions. To establish the claim assume that $c_1 \beta p$ and $c_2 \beta p$, where $c_1 \cap c_2$ is nonempty and so necessarily contains $(s, *, s)$, for some state s of T. Then $c_1, c_2 \in M_c(s)$, where $M_c(s)$ is a reachable marking of $an_c(T)$, by lemma 77(ii). Now, by the fact observed above, we conclude $c_1 = c_2$, justifying the claim.

Thus we can define a relation φ between conditions of $an_0(T)$ and N by taking φ^{op} as the partial function which, for p a condition of N, yields

$$\varphi^{op}(p) = \begin{cases} \bigcup\{c \mid c\beta p\} & \text{if nonempty,} \\ \text{undefined} & \text{otherwise} \end{cases}$$

—as remarked earlier, the union of a nonempty disjoint family of conditions is a condition. That $(\varphi, \eta) : an_0(t) \to N$ is a morphism in **N** follows straightforwardly from $(\beta, \eta) : an_c(T) \to N$ being a morphism in **Safe**. In order for the above diagram to commute we require $\beta = \varphi \circ \gamma$, *i.e.* ,

$$\begin{aligned} c\beta p &\Leftrightarrow c(\varphi \circ \gamma)p \\ &\Leftrightarrow \exists b.\ c\gamma b \ \&\ b\varphi p \\ &\Leftrightarrow c \in \text{conn}(\varphi^{op}(p)) \text{ with } \varphi^{op}(p) \text{ defined.} \end{aligned}$$

But this shows that φ is determined uniquely. $\square$

Corollary 81 *The functors $an_c : \mathbf{A}_c \to \mathbf{Safe}$ and $na_c : \mathbf{Safe} \to \mathbf{A}_c$ form a coreflection with an_c left adjoint to na_c.*

Proof: The natural bijection required for the adjunction follows by composing the bijections of the two adjunctions from $\mathbf{A}_c$ to $\mathbf{A}_0$ with right adjoint V, and from $\mathbf{A}_0$ to $\mathbf{N}$ with left and right adjoint an_0, na_0 respectively, with the bijection of the previous lemma 80; letting T be an object in $\mathbf{A}_c$ and N in **Safe**, there is a chain of bijections between morphisms:

$$T \to na_c(N) = V \circ na_0(N) \text{ in } \mathbf{A}_c,$$

$$T \to na_0(N) \text{ in } \mathbf{A}_0,$$

$$an_0(T) \to N \text{ in } \mathbf{N}, \text{ and}$$

$$an_c(T) \to N \text{ in } \mathbf{Safe}.$$

That the adjunction from $\mathbf{A}_c$ to **Safe**, with left and right adjoints an_c, na_c respectively, is a coreflection follows from lemma 77. $\square$

So certain asynchronous transition systems are again in coreflection with a category of safe nets, but this time with a definition of morphism different from that in section 9.1. The neutral position of asynchronous transition systems with respect to which definition of morphism on nets is taken, argues for their central role as models for concurrency. The coreflection from event structures to asynchronous transition systems cuts down to one $\mathbf{E} \hookrightarrow \mathbf{A}_c$. It composes with that to safe nets to yield a coreflection

$$\mathbf{E} \hookrightarrow \mathbf{A}_c \hookrightarrow \mathbf{Safe}.$$

The composite left adjoint is the construction of an "occurrence net" from an event structure given in [64] (with the addition of a solitary marked condition)—see [97].

11 Semantics

In this section we show how to extend the models to include labels so that they can be used in giving semantics to process languages such as that of section 3. The denotational semantics involves a use of direct limits to handle recursively defined processes. The direct limits are with respect to embedding morphisms in the various categories. In many cases they can be replaced by a simpler treatment based on inclusion morphisms. We conclude by giving an operational semantics equivalent to a denotational semantics using labelled asynchronous transition systems. As will be seen, the operational semantics is obtained by expanding the rules of section 3.2, which generate the transitions, to include extra rules which express the independence between transitions.

11.1 Embeddings

The non-interleaving models, nets, asynchronous transition systems, trace languages and event structures support recursive definitions. The idea of one process approximating another is caught in the notion of an embedding, a suitable kind of monomorphism with respect to which the categorical operations we have seen are continuous, in the sense of preserving ω-colimits. This means that solutions of recursive definitions can be constructed as described for instance in [4]. Recall the least fixed point $fixF$ of a continuous functor $F : \mathbf{X} \to \mathbf{X}$, on a category $\mathbf{X}$ with all ω-colimits and initial object I, is constructed as the colimit of

$$I \xrightarrow{!} F(I) \xrightarrow{F(!)} F^2(I) \xrightarrow{F^2(!)} \cdots \xrightarrow{F^{(n-1)}(!)} F^n(I) \xrightarrow{F^n(!)} \cdots$$

where the morphism $! : I \to F(I)$ is determined uniquely by the initiality of I.

In fact, for all models but nets, it suffices to restrict to inclusion-embeddings, embeddings based on inclusions, which form a large complete partial order. Fortunately the embeddings appropriate for different models are all related to each other. In the case of event structures the embeddings have already been introduced and studied by Kahn and Plotkin under the name *rigid embeddings* (see [40, 97]).

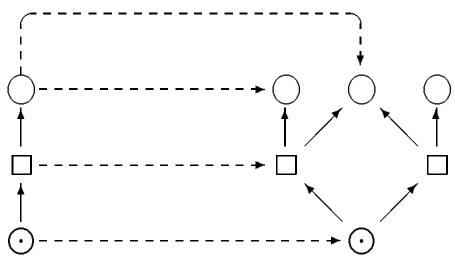
Petri nets: We first consider *embeddings* between nets. These are simply monomorphisms in the category **N**.

Definition: An *embedding* of nets consists of a morphism of nets

$$(\beta, \eta) : N_0 \to N_1$$

such that η is an injective function and β^{op} is surjective, in the sense that for any condition b_0 of N_0 there is a condition b_1 of N_1 for which $b_0 \; \beta \; b_1$.

Example: Injection functions of a sum such as



are examples of embeddings between nets. The need to include such injections, is a chief reason for allowing that part of a net-morphism which relates conditions to not be injective. (Note too there is no "projection morphism" sending e_1 to e_0 and e_2 to undefined.)

Net embeddings are complete with respect to ω-colimits. They have an initial object the net consisting of a simple marked conditions (which coincides with the initial object in the fuller category **N**). The existence of ω-colimits is shown explicitly in the following construction:

Proposition 82 *Let*

$$N_0 \xrightarrow{(\beta_1,\eta_1)} N_1 \xrightarrow{(\beta_2,\eta_2)} \cdots \xrightarrow{(\beta_k,\eta_k)} N_k \xrightarrow{(\beta_{k+1},\eta_{k+1})} \cdots \qquad (\dagger)$$

be an ω-chain of embeddings between nets $N_k = (B_k, M_k, E_k, pre_k, post_k)$, for $k \in \omega$.

Define $N = (B, M, E, pre, post)$ where:

- *B consists of ω-sequences*

$$(b_0, b_1, \ldots, b_k, \ldots)$$

where $b_k \in B_k \cup \{*\}$ such that $b_k = \beta_{k+1}^{op}(b_{k+1})$ for all $k \in \omega$, with the property that $b_m \in B_m$ for some $m \in \omega$; the initial marking M consists of all such sequences for which $b_0 \in M_0$.

- *E consists of ω-sequences*

$$(e_0, e_1, \ldots, e_k, \ldots)$$

where $e_k \in E_k \cup \{*\}$ such that $e_k \neq *$ implies $\eta_{k+1}(e_k) = e_{k+1}$ for all $k \in \omega$, with the property that $e_m \in E_m$ for some $m \in \omega$.

- *the maps $pre : E \to \mathcal{P}ow(B)$ and $post : E \to \mathcal{P}ow(B)$ satisfy*

$$\begin{aligned} b \in pre(e) &\Leftrightarrow \forall k \in \omega.\ (e_k \neq * \Rightarrow (b_k \neq * \ \&\ b_k \in pre_k(e_k))) \\ b \in post(e) &\Leftrightarrow \forall k \in \omega.\ (e_k \neq * \Rightarrow (b_k \neq * \ \&\ b_k \in post_k(e_k))), \end{aligned}$$

where we use e_k and b_k for the k-th components of the sequences e and b respectively.

Then N is a net. For each $k \in \omega$, the pair $f_k = (\gamma_k, \epsilon_k)$ consisting of a relation $\gamma_k \subseteq B \times B_k$ such that

$$c \gamma_k b \Leftrightarrow c = b_k$$

and a function $\epsilon_k : E_k \to E$ such that

$$\epsilon_k(e') = e \Leftrightarrow e' = e_k$$

is an embedding of nets $f_k : N_k \to N$. Furthermore, N and the collection of embeddings f_k, $k \in \omega$, is a colimit of the ω-chain (†).

Asynchronous transition systems: An embedding between asynchronous transition systems consists of a monomorphism which reflects the independence relation.

Definition: An *embedding* of asynchronous transition systems consists of a morphism

$$(\sigma, \eta) : T_0 \to T_1,$$

between asynchronous transition systems T_0 and T_1 with independence relations I_0, I_1 respectively, such that σ and η are injective and

$$\eta(e_0), \eta(e_1) \text{ defined } \& \ \eta(e_0) I_1 \eta(e_1) \Rightarrow e_0 \ I_0 \ e_1$$

for any events e_0, e_1 of T_0.

Proposition 83
(i) If $f : N_0 \to N_1$ is an embedding of nets, then $na(f) : na(N_0) \to na(N_1)$ is an embedding of asynchronous transition systems. Moreover, na preserves ω-colimits of embeddings.
(ii) If $g : T_0 \to T_1$ is an embedding of asynchronous transition systems, then $an(g) : an(T_0) \to an(T_1)$ is an embedding of nets. Moreover, an preserves ω-colimits of embeddings.

The operations on asynchronous transition systems we have seen are all continuous with respect to an order based on embeddings which are inclusions:

Definition: Let $T_0 = (S_0, i_0, E_0, I_0, tran_0)$ and $T_1 = (S_1, i_1, E_i, I_i, tran_1)$ be asynchronous transition systems. Define $T_0 \trianglelefteq T_1$ iff $S_0 \subseteq S_1, E_0 \subseteq E_1$ and (σ, η) is an embedding where σ is the inclusion $S_0 \hookrightarrow S_1$ and η the inclusion $E_0 \hookrightarrow E_1$.

Asynchronous transition systems have ω-colimits of embeddings. In particular, if

$$T_0 \trianglelefteq \cdots \trianglelefteq T_n \trianglelefteq \cdots$$

is an ω-claim of asynchronous transition systems $T_n = (S_n, i_n, E_n, I_n, tran_n)$, it has a least upper bound

$$(\bigcup_{n\in\omega} S_n, i_0, \bigcup_{n\in\omega} E_n, \bigcup_{n\in\omega} I_n, \bigcup_{n\in\omega} tran_n)$$

which is not only an ω-colimit in the category of inclusion-embeddings, but also in the category of embeddings. The situation restricts to asynchronous transition systems in $\mathbf{A}_0$; they are closed under least upper bounds of ω-chains under $\trianglelefteq$.

Trace languages: Embeddings on asynchronous transition systems induce embeddings on trace languages via the identification given by *tla*:

Definition: An *embedding* of trace languages consists of a morphism $\eta : T \to T'$ of trace languages T, T', with independence relations I, I' respectively, such that η is injective and

$$\eta(a), \eta(b) \text{ defined } \ \& \ \eta(a) I' \eta(b) \Rightarrow aIb, \quad \text{for all } a, b \in E.$$

Let $T = (M, E, I), T' = (M', E', I')$ be trace languages. Define $T \trianglelefteq T'$ iff

$$\begin{aligned} & M \subseteq M', \\ & E \subseteq E' \text{ and} \\ & aIb \Leftrightarrow aI'b, \quad \text{for all } a, b \in E. \end{aligned}$$

Again, embeddings and inclusion-embeddings have colimits of ω-chains which in the case of inclusion embeddings are given by unions. The functors *atl* and *tla* are continuous with respect to inclusion-embeddings.

Event structures: To treat recursively defined event structures we use a notion of embedding equivalent to that of the *rigid embeddings* of Kahn and Plotkin (see [40, 97]). Note that in the case of event structures (though not for the other models of this section) embeddings are always associated with projection morphisms in the opposite direction. When the embeddings are inclusions they amount to a substructure relation on event structures.

Definition: An *embedding* of event structures consists of a morphism $\eta : ES_0 \to ES_1$ between event structures ES_0, ES_1 where η is injective and such that its opposite, the partial function η^{op}, is a morphism of event structures $\eta^{op} : ES_1 \to ES_0$.

Let $ES_0 = (E_0, \leq_0, \#_0), ES_1 = (E_1, \leq_1, \#_1)$ be event structures. Define $ES_0 \trianglelefteq ES_1$ iff

$$E_0 \subseteq E_1,$$

$$\forall e \in E_1.\ e \leq_0 e_0 \Leftrightarrow e \leq_1 e_0,$$

for all $e_0 \in E_0$, and

$$e \#_0 e' \text{ iff } e \#_1 e',$$

for all $e, e' \in E_0$.

The $\trianglelefteq$ order on event structures is a special case of the order on trace languages:

Proposition 84

(i) *If $ES \trianglelefteq ES'$,for event structures ES, ES', then $etl(ES) \trianglelefteq etl(ES')$, for the associated trace languages. Moreover, etl preserves ω-colimits of inclusion-embeddings.*

(ii) *If $T \trianglelefteq T'$, for trace languages T, T', then $tle(T) \trianglelefteq tle(T')$, for the associated event structures. Moreover, tle preserves ω-colimits of inclusion-embeddings.*

11.2 Labelled structures

For noninterleaving models of concurrency like event structures, we distinguish between events, which carry the independence structure, and labels of the kind one sees in process algebras, whose use is to specify the nature of events, to determine for example how they synchronise. The denotation of a process, for example from the process language **Proc**, will most naturally be a labelled structure. The models we consider possess a set of events to which we can attach a labelling function. The set of events of an object X in a typical category $\mathbf{X}$ of structures (for example, $\mathbf{X}$ could be the category of event structures) is given by a functor $E : \mathbf{X} \to \mathbf{Set}_*$. This permits us to adjoin labelling sets to several different categories of models in the same way, using the following construction:

Definition: Let $E : \mathbf{X} \to \mathbf{Set}_*$ be a functor from a category $\mathbf{X}$. Define $\mathcal{L}(\mathbf{X})$ to be the category consisting of

> *objects* $(X, l : E(X) \to L)$ where X is an object of $\mathbf{X}$ and l is a morphism in $\mathbf{Set}$, and
>
> *morphisms* pairs $(f, \lambda) : (X, l : E(X) \to L) \to (X', l' : E(X') \to L')$ where $f : X \to X'$ in $\mathbf{X}$ and $\lambda : L \to L'$ in $\mathbf{Set}$ satisfy
>
> $$l' \circ E(f) = \lambda \circ l,$$
>
> with composition defined coordinatewise, *i.e.* $(f', \lambda') \circ (f, \lambda) = (f' \circ f, \lambda' \circ \lambda)$ provided $f' \circ f$ and $\lambda' \circ \lambda$ are defined.

To understand how this construction is used, take $\mathbf{X}$ to be one kind of model, say event structures, so $\mathbf{X}$ is $\mathbf{E}$. Then understanding E to be the forgetful functor to sets of events and partial functions, has the effect of adjoining to event structures extra structure in the form of total labelling functions on events: the objects of the category $\mathcal{L}(\mathbf{E})$ are labelled event structures $(ES, l : E \to L)$ where ES is an event structure and l is a total function from its events E to a set of labels L; morphisms $(ES, l : E \to L) \to (ES', l' : E' \to L')$ are pairs (η, λ), with $\eta : ES \to_* ES'$ a morphism of event structures, and $\lambda : L \to_* L'$ such that $l' \circ \eta = \lambda \circ l$.

Products and coproducts in $\mathcal{L}(\mathbf{E})$ are obtained from the corresponding constructions in the unlabelled category because of the following general facts:

Proposition 85 *Let $E : \mathbf{X} \to \mathbf{Set}_*$ be a functor from a category $\mathbf{X}$. Assume $\mathbf{X}$ has products. Then, a product of $(X_0, l_0 : E(X_0) \to L_0)$ and $(X_1, l_1 : E(X_1) \to L_1)$ in $\mathcal{L}(\mathbf{X})$ is given by $(X, l : E(X) \to L)$ with projections (η_0, λ_0), (η_1, λ_1), where*

- *X is a product of X_0, X_1 in $\mathbf{X}$ with projections $\eta_0 : X \to X_0, \eta_1 : X \to X_1$*
- *L is a product of L_0, L_1 in $\mathbf{Set}_*$ with projections $\lambda_0 : L \to L_0, \lambda_1 : L \to L_1$*
- *$l = \langle l_0 \circ E(\eta_0), l_1 \circ E(\eta_1) \rangle : E(X) \to L$ is the unique mediating morphism to the product L such that $\lambda_0 \circ l = l_0 \circ E(\eta_0)$ and $\lambda_1 \circ l = l_1 \circ E(\eta_1)$.*

Proposition 86 *Let $E : \mathbf{X} \to \mathbf{Set}_*$ be a functor from a category $\mathbf{X}$. Assume $\mathbf{X}$ has coproducts preserved by E. Then, a coproduct of $(X_0, l_0 : E(X_0) \to L_0)$ and $(X_1, l_1 : E(X_1) \to L_1)$ in $\mathcal{L}(\mathbf{X})$ is given by $(X, l : E(X) \to L)$ with injections (η_0, λ_0), (η_1, λ_1), where*

- *X is a coproduct of X_0, X_1 in $\mathbf{X}$ with injections $\eta_0 : X_0 \to X, \eta_1 : X_1 \to X$*
- *L is a coproduct of L_0, L_1 in $\mathbf{Set}_*$ with injections $\lambda_0 : L_0 \to L, \lambda_1 : L_1 \to L$*
- *$l = [\lambda_0 \circ l_0, \lambda_1 \circ l_1] : E(X) \to L$ is the unique mediating morphism from the coproduct $E(X)$ such that $\lambda_0 \circ l_0 = l \circ E(\eta_0)$ and $\lambda_1 \circ l_1 = l \circ E(\eta_1)$.*

There is a functor $p : \mathcal{L}(\mathbf{X}) \to \mathbf{Set}_*$; a morphism of labelled structures

$$(f, \lambda) : (X, l : E(X) \to L) \to (X', l' : E(X') \to L')$$

is sent to

$$\lambda : L \to_* L'.$$

For any total function $\lambda : L \to L'$ in $\mathbf{Set}_*$, this functor does have a strong cocartesian lifting of λ with respect to any object $(X, l : E(X) \to L)$ in $\mathcal{L}(\mathbf{X})$: it is given by the morphism

$$(1_X, \lambda) : (X, l : E(X) \to L) \to (X, \lambda \circ l : E(X) \to L')$$

in $\mathcal{L}(\mathbf{X})$. This yields a relabelling operation when $\mathbf{X}$ is specialised to one of the models.

For any of the models, there are also strong cartesian liftings of inclusions $L \hookrightarrow L'$ in $\mathbf{Set}_*$ with respect to a labelled structure $(X, l : E(X) \to L)$, though this requires an argument resting on the fact that the categories of structures (without labels) that we consider support an operation of restriction to a prescribed subset of events. For example, given an event structure $ES = (E', \leq', \#')$ and a specified subset $E \subseteq E'$ there is an event structure, the *restriction* of ES to E gives an event structure $(E_0, \leq, \#)$ as follows:

its set of events consists of $E_0 = \{e \in E \mid \forall e' \leq e. e' \in E\}$;

its causal dependency relation satisfies

$$e \leq e' \Leftrightarrow e, e' \in E_0 \;\&\; e \leq' e';$$

its conflict relation satisfies

$$e \# e' \Leftrightarrow e, e' \in E_0 \;\&\; e \#' e'.$$

To treat recursion on labelled structures we extend embeddings to labelled structures, such as $\mathcal{L}(\mathbf{E})$. A morphism of labelled structures

$$(f, \lambda) : (X, l : E \to L) \to (X', l' : E' \to L')$$

is taken to be an embedding (or an inclusion-embedding) if $f : X \to X'$ is an embedding (or an inclusion embedding) and λ is an inclusion of sets. The labelled structures have colimits of ω-chains formed from colimits of the unlabelled structures. In particular, a chain

$$(X_0, l_0) \trianglelefteq \cdots \trianglelefteq (X_n, l_n) \trianglelefteq \cdots$$

of labelled structures $(X_n, l_n : E_n \to L_n)$, has least upper bound $(\bigcup_{n \in \omega} X_n, \bigcup_{n \in \omega} l_n)$. The labelling function $\bigcup_{n \in \omega} l_n$ has domain $\bigcup_{n \in \omega} E_n$ and codomain $\bigcup_{n \in \omega} L_n$.

In section 3 we had to go to a little trouble to extend the restriction and relabelling operations to all transition systems regardless of their labelling set. In general, some care is needed in making functors with respect to embeddings out of some of the operations. The operations of restriction and relabelling $(- \restriction \Lambda)$ and $(-\{\Xi\})$ yield functors on categories of embeddings. Suppose there is an embedding $f : X \to X'$ between labelled structures X, X' with labelling sets L, L' respectively, necessarily related by an inclusion $L \hookrightarrow L'$. The structure X with labelling set L restricts to $X \restriction \Lambda$ associated with a particular cartesian lifting

$$c : X \restriction \Lambda \to X$$

of the inclusion $L \cap \Lambda \hookrightarrow L$. Similarly, X' is associated with the cartesian lifting

$$c' : X' \restriction \Lambda \to X'$$

of the inclusion $L' \cap \Lambda \hookrightarrow L'$. Because c' is strong cartesian there is a unique morphism

$$(f \restriction \Lambda) : X \restriction \Lambda \to X' \restriction \Lambda$$

projecting to the inclusion $L \cap \Lambda \hookrightarrow L' \cap \Lambda$ such that $f \circ c = c' \circ (f \restriction \Lambda)$. This ensures that $(- \restriction \Lambda)$ is a functor from the subcategory with embeddings. Moreover, for each model, $(f \restriction \Lambda)$ is an (inclusion-)embedding provided f is. In a similar way a relabelling function Ξ is associated with cocartesian liftings $X \to X\{\Xi\}$ of $L \to \Xi L$ for any structure X with labelling set L, and gives rise to a functor with respect to embeddings. For all the models here, it is a straightforward matter to define a prefixing operation on the various labelled structures so that it is continuous with respect to a choice of embedding given. The labelled versions of continuous functors are continuous.

The various categories of labelled structures, such as $\mathcal{L}(\mathbf{E})$ for example, provide a semantics to the process language **Proc** interpreting constructions in the process language as the appropriate universal construction, so abstractly this proceeds exactly as in section 3.

11.3 Operational semantics

11.3.1 Transition systems with independence

The model of asynchronous transition systems is based on events which carry an independence relation. The nature of these events can then be specified by a further level of labelling. There is an alternative, more direct, presentation of (certain kinds of) labelled asynchronous transition systems, got by extending transition sytems with an independence relation on its transitions. Transition systems with independence are definable by the techniques of structural operational semantics in a way which directly extends that of section 3.

Definition: A *transition system with independence*[10] is defined to be a structure

$$(S, i, L, \mathit{Tran}, I)$$

where (S, i, L, Tran) is a transition system and the independence relation $I \subseteq \mathit{Tran}^2$ is an irreflexive, symmetric relation, such that

(1) $(s, a, s_1) \sim (s, a, s_2) \Rightarrow s_1 = s_2$

(2) $(s, a, s_1) I (s, b, s_2) \Rightarrow \exists u.\ (s, a, s_1) I (s_1, b, u)\ \&\ (s, b, s_2) I (s_2, a, u)$

(3) $(s, a, s_1) I (s_1, b, u) \Rightarrow \exists s_2.\ (s, a, s_1) I (s, b, s_2)\ \&\ (s, b, s_2) I (s_2, a, u)$

(4)

$$\text{(i)}(s, a, s_1) \prec (s_2, a, u) I (w, b, w') \Rightarrow (s, a, s_1) I (w, b, w')$$

$$\text{(ii)}(w, b, w') I (s, a, s_1) \prec (s_2, a, u) \Rightarrow (w, b, w') I (s_2, a, u)$$

where the relation $\prec$ between transitions is defined by

$$(s, a, s_1) \prec (s_2, a, u) \Leftrightarrow \exists b.\ (s, a, s_1) I (s, b, s_2)\ \&\ (s, a, s_1) I (s_1, b, u)\ \&\ (s, b, s_2) I (s_2, a, u),$$

and $\sim$ is the least equivalence relation including $\prec$.

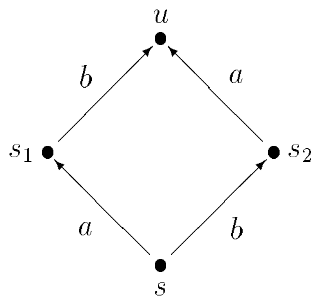
As morphisms on transition systems with independence we take morphisms on the underlying transition systems which preserve independence, *i.e.* a morphism $(\sigma, \lambda) : T \to T'$ should satisfy

> If (s, a, s') and (u, b, u') are independent transitions of T and $\lambda(a)$ and $\lambda(b)$ are both defined, then $(\sigma(s), \lambda(a), \sigma(s'))$ and $(\sigma(u), \lambda(b), \sigma(u'))$ are independent transitions of T'.

[10] Axiom 2 of transition systems with independence is not essential to much of the development. It ensures that the trace language of a transition system with independence is coherent, so that the associated event structure has the property that conflict is determined in a binary fashion.

Composition is inherited from that in **T**. We write **TI** for the category of transition systems with independence.

Thus transition systems with independence are precisely what their name suggests, *viz.* transition systems of the kind used to model languages like CCS and CSP but with an additional relation expressing when one transition is independent of another. The axioms (2) and (3) describe intuitive properties of independence, similar to those we have seen. The relation $\prec$ expresses when two transitions represent occurrences of the same event. This relation extends to an equivalence relation $\sim$ between transitions; the equivalence classes $\{(s,a,s')\}_\sim$, of transitions (s,a,s'), are the events of the transition system with independence. Property (4) is then seen as asserting that the independence relation respects events. Note that property (4) implies that if $(s,a,s_1) \prec (s_2,a,u)$, *i.e.* there is a "square" of transitions



with

$$(s,a,s_1)I(s,b,s_2) \ \&\ (s,a,s_1)I(s_1,b,u) \ \&\ (s,b,s_2)I(s_2,a,u),$$

then we also have the independence

$$(s_1,b,u)I(s_2,a,u).$$

The first property (1) simply says that the occurrence of an event at a state yields a unique state. Note that property (1) implies the uniqueness of the states, u and s_2, whose existence is asserted by properties (2) and (3) respectively.

In this way a transition system with independence can be viewed as an asynchronous transition system in which the events are labelled, an event $\{(s,a,s')\}_\sim$ carrying the label a. The resulting asynchronous transition system is *extensional* in that it has the property that

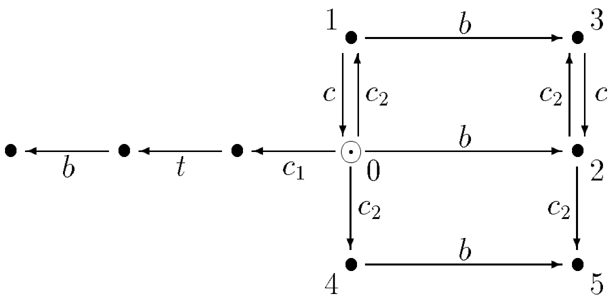
(1) for any label there is at most one event with that label involved in a transition between two states.

It is special in another way too. An asynchronous transition system can be regarded as a transition system with independence, in which the independence on transitions is induced by that on its events. The asynchronous transition systems which result from transition systems with independence have the special property that

(2) the map $\{(s, e, s')\}_\sim \mapsto e$ is a bijection.

There is in fact an equivalence between the category of transition systems with independence and the full subcategory of $\mathcal{L}(\mathbf{A})$ for which the objects are labelled transition systems with the properties (1) and (2).

Let's return to our example of 3.3. It is now an easy matter to extend the transition-system semantics there to take account of independence. We simply specify which transitions are independent of which others. Copying the transition system of 3.3,



we assert in addition the following independencies

$$(0, c_2, 1)I(0, b, 2), \quad (1, c, 0)I(1, b, 3), \quad (0, b, 2)I(0, c_2, 4)$$

which then generate others by the axioms in the definition of a transition system with independence.

11.3.2 Operational rules

Transition systems with independence have the feature that they are definable by structural operational semantics in much the same way as transition systems, but with the usual rules for transitions being supplemented by rules specifying the independence relation between transitions.

To motivate the rules we first examine how the product lends itself readily to a presentation via rules of structural operational semantics. Assume $T_0 = (S_0, i_0, L_0, \mathit{Tran}_0, I_0)$ and $T_1 = (S_1, i_1, L_1, \mathit{Tran}_1, I_1)$ are transition systems with independence. Their categorical product $T_0 \times T_1$ is $(S, i, L, \mathit{Tran}, I)$ where (S, i, L, Tran) is the product of the underlying transition systems $(S_0, i_0, L_0, \mathit{Tran}_0)$, $(S_1, i_1, L_1, \mathit{Tran}_1)$, with projections $(\rho_0, \pi_0), (\rho_1, \pi_1)$, and the independence relation I on transitions is given by

$$\begin{aligned}
&(s, a, s')I(u, b, u') \text{ iff} \\
&\pi_0(a), \pi_0(b) \text{ defined } \Rightarrow (\rho_0(s), \pi_0(a), \rho_0(s'))I_0(\rho_0(u), \pi_0(b), \rho_0(u')) \ \& \\
&\pi_1(a), \pi_1(b) \text{ defined } \Rightarrow (\rho_1(s), \pi_1(a), \rho_1(s'))I_1(\rho_1(u), \pi_1(b), \rho_1(u')).
\end{aligned}$$

The characterisation of the independence relation can be simplified through the use of idle transitions. An independence relation like $I \subseteq \mathit{Tran} \times \mathit{Tran}$ extends to a relation $I_* \subseteq \mathit{Tran}_* \times \mathit{Tran}_*$ in which

$$(s, a, s')I_*(u, b, u') \Leftrightarrow a = * \text{ or } b = * \text{ or } (s, a, s')I(u, b, u').$$

An idle transition is thus always independent of any transition, idle or otherwise. Now we have the simplification:

$$\begin{array}{l}(s, a, s')I_*(u, b, u') \;\text{ iff}\\ (\rho_0(s), \pi_0(a), \rho_0(s'))I_{0*}(\rho_0(u), \pi_0(b), \rho_0(u')) \;\&\\ (\rho_1(s), \pi_1(a), \rho_1(s'))I_{1*}(\rho_1(u), \pi_1(b), \rho_1(u')).\end{array}$$

We have already seen rules to give the transitions of the product (section 3.2). To define the product of transition systems with independence we adjoin the following rule, which reformulates the condition for two transitions of a product to be independent:

$$\frac{(s_0, a_0, s_0')I_{0*}(u_0, b_0, u_0'), \qquad (s_1, a_1, s_1')I_{1*}(u_1, b_1, u_1')}{((s_0, s_1), a_0 \times a_1, ({s'}_0, {s'}_1))I_*((u_0, u_1), b_0 \times b_1, ({u'}_0, {u'}_1))}$$

Similarly, the fibre coproduct of transition systems with independence is given by the fibre coproduct of the underlying transition systems together with an independence relation inherited directly from the components. This too can be expressed by simple rules, which are essentially unchanged in the nondeterministic sum $\oplus$, where we first enlarge the labelling sets to their union and then form the fibre coproduct. Let T_0 and T_1 be the transition systems with independence above. Their sum $T_0 \oplus T_1$ consists of a transition system, formed as the nondeterministic sum of their underlying transition systems associated with injection functions in_0, in_1 on states, together with an independence relation satisfying

$$\begin{array}{l}(s, a, s')I(u, b, u')\\ \;\text{iff}\\ [\exists s_0, s_0', u_0, u_0'.\\ s = in_0(s_0) \;\&\; s' = in_0(s_0') \;\&\; u = in_0(u_0) \;\&\; u' = in_0(u_0') \;\&\; (s_0, a, s_0')I_0(u_0, b, u_0')]\\ \;\text{or}\\ [\exists s_1, s_1', u_1, u_1'.\\ s = in_1(s_1) \;\&\; s' = in_1(s_1') \;\&\; u = in_1(u_1) \;\&\; u' = in_1(u_1') \;\&\; (s_1, a, s_1')I_1(u_1, b, u_1')].\end{array}$$

Expressed by rules the condition on the independence relation becomes:

$$\frac{(s_0, a, s_0')I_0(u_0, b, u_0')}{(in_0(s_0), a, in_0(s_0'))I(in_0(u_0), b, in_0(u_0'))} \qquad \frac{(s_1, a, s_1')I_1(u_1, b, u_1')}{(in_1(s_1), a, in_1(s_1'))I(in_1(u_1), b, in_1(u_1'))}$$

As usual a restriction can be understood as a cartesian lifting of an inclusion morphism on labelling sets; there is an obvious functor from **TI** to $\mathbf{Set}_*$ projecting to the labelling sets and the labelling functions between them. Letting $T = (S, i, L, \mathit{Tran}, I)$ be a transition system with independence, the restriction to a subset of labels Λ is

$$T \upharpoonright \Lambda = (S, i, L', \mathit{Tran}', I')$$

where $L' = L \cap \Lambda$, $\mathit{Tran}' = \mathit{Tran} \cap (S \times L' \times S)$ and $I' = I \cap (\mathit{Tran}' \times \mathit{Tran}')$. Although this operation may change the $\sim$ relation, increasing the number of events, it preserves the axioms required of a transition system with independence. The rule in the operational semantics for the independence relation of a restriction expresses that it is got simply by cutting down the original independence relation.

Relabelling is associated with a cocartesian lifting of the relabelling function on labelling sets. In defining it we can take advantage of a *unicity property* of those transition systems arising from the operational semantics:

> Suppose $s \xrightarrow{a} s'$ and $s \xrightarrow{b} s'$ are transitions obtained from the operational semantics (version 2) of section 3.2. Then $a = b$.

This property is easily observed to be preserved by the rules.
Let $T = (S, i, L, \mathit{Tran}, I)$ be a transition system with independence, assumed to satisfy the unicity property

$$(s, a, s') \in \mathit{Tran} \;\&\; (s, b, s') \in \mathit{Tran} \Rightarrow a = b.$$

For $\Xi : L \to L'$ the relabelling

$$T\{\Xi\} = (S, i, L', \mathit{Tran}', I'),$$

where $\mathit{Tran}' = \{(s, b, s') \mid \exists a.\ b = \Xi(a) \;\&\; (s, a, s') \in \mathit{Tran}\}$ and

$$(s, a, s') I' (t, b, t') \Leftrightarrow \exists a', b'.\ a = \Xi(a') \;\&\; b = \Xi(b') \;\&\; (s, a', s') I (t, b', t').$$

Because the transition system T satisfies the unicity property the construction $T\{\Xi\}$ yields a transition system with independence (without the assumption of unicity the new relation I', as defined, need not respect events, and a more complicated definition is needed). Consequently in the operational semantics we can get away with a rule which says the independence relation of the relabelled transition system is simply the image of the original.

We obtain an operational semantics for **Proc** as transition system with independence by extending version 2 of the rules of section 3 for the transitions between (tagged) states by the following rules for the independence relation (also relating idle transitions):

Rules for independence

$$(s,a,s')\ I\ (u,*,u) \qquad\qquad \frac{(s,a,s')\ I\ (u,b,u')}{(u,b,u')\ I\ (s,a,s')}$$

$$\frac{(s,a,s')\ I\ (t,b,t')}{((n,s),a,(n,s'))\ I\ ((n,t),b,(n,t'))}$$

Sum:

$$\frac{(s,a,s')\ I\ (s,b,s'')}{(s\oplus t,a,(0,s'))\ I\ (s\oplus t,b,(0,s''))}\ a,b\neq * \qquad \frac{(t,a,t')\ I\ (t,b,t'')}{(s\oplus t,a,(1,t'))\ I\ (s\oplus t,b,(1,t''))}\ a,b\neq *$$

$$\frac{(s,a,s')\ I\ (u,b,u')}{(s\oplus t,a,(0,s'))\ I\ ((0,u),b,(0,u'))}\ u\not\equiv s, a\neq *$$

$$\frac{(t,a,t')\ I\ (u,b,u')}{(s\oplus t,a,(1,t'))\ I\ ((1,u),b,(1,u'))}\ u\not\equiv t, a\neq *$$

Product:

$$\frac{(s_1,a_1,s_1')\ I\ (s_2,a_2,s_2') \quad (t_1,b_1,t_1')\ I\ (t_2,b_2,t_2')}{(s_1\times t_1,a_1\times b_1,s_1'\times t_1')\ I\ (s_2\times t_2,a_2\times b_2,s_2'\times t_2')}$$

Restriction and relabelling:

$$\frac{(s,a,s')\ I\ (t,b,t')}{(s\lceil\Lambda,a,s'\lceil\Lambda)\ I\ (t\lceil\Lambda,b,t'\lceil\Lambda)} a,b\in\Lambda \qquad \frac{(s,a,s')\ I\ (t,b,t')}{(s\{\Xi\},\Xi(a),s'\{\Xi\})\ I\ (t\{\Xi\},\Xi(b),t'\{\Xi\})}$$

Recursion:

$$\frac{(t[rec\,x.t/x],a,s)\ I\ (t[rec\,x.t/x],b,u)}{(rec\,x.t,a,(2,s))\ I\ (rec\,x.t,b,(2,u))} a,b\neq *$$

$$\frac{(t[rec\,x.t/x],a,s)\ I\ (u,b,u')}{(rec\,x.t,a,(2,s))\ I\ ((2,u),b,(2,u'))}\ u\not\equiv t[rec\,x.t/x], a\neq *$$

A closed term of **Proc** determines a transition system with independence consisting of all those states and transitions forwards-reachable from it together with an independence relation determined by the rules above. Notice there are no extra rules for prefixing because the transition immediately possible for a pre-fixed process is not independent to any other. The rules for product, restriction and relabelling are straightforward reformulations as rules of the requirements on their independence relations. The rules for sum and recursion require further explanation. For a sum $s \oplus t$, taking the injection functions in_0, in_1 on states to satisfy, *e.g.*

$$in_0(s) = s \oplus t, \quad \text{and} \quad in_0(u) = (0, u) \text{ if } u \not\equiv s,$$

we can understand the rules for sum, together with the rule for tagged terms, as saying that independence for a sum is precisely that inherited separately from the components. Because the transition system is acyclic (lemma 10), there is an isomorphism between the transition system reachable from $rec\ x.t$ and its unfolding $t[rec\,x.t/x]$ (this fact is used earlier in the proof of theorem 11). The isomorphism is given by

$$\begin{array}{lll} rec\,x.t & \mapsto & t[rec\,x.t/x] \\ (2, u) & \mapsto & u. \end{array}$$

The rules for recursively defined processes, with the final rule for tagged terms, ensure that transitions reachable from $rec\ x.t$ are independent precisely when their images under this isomorphism are independent.

A denotational semantics where denotations are transition systems with independence can be presented along standard lines; the categorical constructions defined above are used to interpret the operations.

We have already discussed the categorical constructions in **TI** which are used to interpret the operations of the process language. It remains to handle recursion. We define an appropriate ordering, with respect to which all the constructions are continuous:

Definition: Let $T = (S, i, L, tran, I)$ and $T' = (S', i', L', tran', I')$ be transition systems with independence. Define $T \trianglelefteq T'$ iff

$$S \subseteq S' \text{ with } i = i,\ L \subseteq L',\ tran \subseteq tran' \text{ and}$$

$$\forall (s, a, s'), (t, b, t') \in tran.\ (s, a, s')I(t, b, t') \Leftrightarrow (s, a, s')I'(t, b, t').$$

Now, as earlier in section 3, for straightforward transition systems, we can give denotations to recursively defined processes. The result is that with respect to an environment ρ assigning meanings to process variables as transition systems with independence, we can give the denotation of a process term t as a transition system with independence

$$\mathbf{TI}[\![t]\!]\rho.$$

The denotational semantics agrees with the operational semantics. The proof proceeds analogously to that of theorem 11—further details are given in Appendix C(b).

Definition: For $T = (S, i, L, \mathit{Tran}, I)$ consisting of a transition system (S, i, L, Tran) and relation $I \subseteq \mathit{Tran} \times \mathit{Tran}$, define $\mathcal{R}(T)$ to be $(S', i, L', \mathit{Tran}', I')$ consisting of states S' reachable from i, with initial state i, and transitions $\mathit{Tran}' = \mathit{Tran} \cap (S' \times L \times S')$ with labelling set L' consisting of those labels appearing in Tran' and $I' = I \cap (\mathit{Tran}' \times \mathit{Tran}')$.

Assume t is a closed term of **Proc**. Let T consist of the transition system got from version 2 in section 3, with initial state t, with independence relation given by the rules above. Define

$$\mathcal{O}p(t) = \mathcal{R}(T).$$

Theorem 87 *Let t be a closed process term. Then, for any arbitrary environment ρ,*

$$\mathcal{O}p(t) \cong \mathcal{R}(\mathbf{TI}[\![t]\!]\rho),$$

a label-preserving isomorphism.

The denotational semantics in **TI** is closely related to that in $\mathcal{L}(\mathbf{A})$ which we write as $\mathbf{A}[\![t]\!]\rho$, for a term t and an environment ρ interpreting variables in $\mathcal{L}(\mathbf{A})$. There is an obvious functor from $\mathcal{L}(\mathbf{A})$ to **TI** (it is not however adjoint to that functor identifying a transition system with idependence with an equivalent labelled asynchronous transition system). On objects it acts as follows:

Definition: Let $T = (S, i, E, I, \mathit{Tran}, l : E \to L)$ be an object of $\mathcal{L}(\mathbf{A})$. Define $u(T)$ to be $(S, i, L, \mathit{Tran}', I')$ where

$$(s, a, s') \in \mathit{Tran}' \Leftrightarrow \exists e.\ l(e) = a \ \&\ (s, e, s') \in \mathit{Tran}$$
$$(s, a, s')I'(t, b, t') \Leftrightarrow \exists e_0, e_1.\ l(e_0) = a \ \&\ l(e_1) = b \ \&\ (s, e_0, s')I(t, e_1, t').$$

Theorem 88 *Let t be a term of the process language* **Proc**. *For any environment ρ interpreting process variables in $\mathcal{L}(\mathbf{A})$,*

$$\mathbf{TI}[\![t]\!](u \circ \rho) = u(\mathbf{A}[\![t]\!]\rho).$$

Proof: The operation u can be shown to be continuous with respect to the orderings $\trianglelefteq$ and to preserve the operations of **Proc**. A structural induction on terms t of **Proc** shows that

$$\mathbf{TI}[\![t]\!](u \circ \rho) = u(\mathbf{A}[\![t]\!]\rho),$$

for an environment ρ interpreting variables in $\mathcal{L}(\mathbf{A})$. The case where t is a recursive process relies on the fact that if F and G are continuous functions on (large) cpo's **TI** and $\mathcal{L}(\mathbf{A})$ respectively, ordered by $\trianglelefteq$, such that

$$F \circ u = u \circ G$$

then, because u is continuous and preserves the bottom element in the definitions by recursion,

$$fixF = u(fixG).$$

$\square$

As we will see, the coreflections between categories of unlabelled structures extend to categories of labelled structures. In particular, this yields a coreflection $\mathcal{L}(\mathbf{E}) \hookrightarrow \mathcal{L}(\mathbf{A})$. This coreflection cuts down to one $\mathcal{L}(\mathbf{E}) \hookrightarrow \mathbf{TI}$ and semantics in $\mathcal{L}(\mathbf{A})$ and $\mathbf{TI}$ unfold to the same semantics in labelled event structures.[11]

12 Relating models

Earlier in section 11.2, it was seen how to attach labels to events of structures in a uniform way. In relating semantics in terms of the different models, we shall also wish to extend functors between categories of models to functors between their labelled versions. For this we use the fact that the functors of interest are accompanied by natural transformations, so that a general scheme described in the following definition applies. The components of the natural transformation relate the event sets before and after application of the functor; for example, the natural transformation accompanying the functor from trace languages to event structures has components mapping events of a trace language to their associated symbols in the alphabet.

Definition: Let $E_C : \mathbf{C} \to \mathbf{Set}_*$ and $E_D : \mathbf{D} \to \mathbf{Set}_*$ be functors (taking structures to their underlying event sets).

Suppose $F : \mathbf{C} \to \mathbf{D}$ is a functor and $\phi : E_D \circ F \to E_C$ is a natural transformation with components in $\mathbf{Set}$ (the natural transformation relates the event sets resulting from the application of F to those originally). Define the functor $\mathcal{L}(F, \phi) : \mathcal{L}(\mathbf{C}) \to \mathcal{L}(\mathbf{D})$ to act on objects so

$$(C, l : E_C(C) \to L) \mapsto (F(C), l \circ \phi_C : E_D \circ F(C) \to L)$$

and on morphisms so

$$(f, \lambda) \mapsto (F(f), \lambda)$$

where $(f, \lambda) : (C, l : E_C(C) \to L) \to (C', l' : E_C(C') \to L')$.

[11] It follows that the labelled event structure obtained from the Petri net semantics is isomorphic to that got by "unfolding" the operational semantics. We can also ask about the following method for obtaining a labelled-net semantics directly from the semantics in **TI**. Certainly we can regard a transition system with independence as a labelled asynchronous transition system (how, is explained early in this section) and thus we can obtain a net via the adjunction between asynchronous transition systems and nets. At the time of writing, it is not decided whether or not this yields an asynchronous transition system in $\mathbf{A}_0$, and thus, via the coreflection, a net with the same underlying asynchronous transition system as its behaviour.

Under reasonable conditions the labelling operation $\mathcal{L}(-)$ preserves adjunctions, coreflections and reflections:

Lemma 89 *Let $E_C : \mathbf{C} \to \mathbf{Set}_*$ and $E_D : \mathbf{D} \to \mathbf{Set}_*$ be functors.*

Suppose $F : \mathbf{C} \to \mathbf{D}$ is a functor and $\phi : E_D \circ F \to E_C$ is a natural transformation. Suppose $G : \mathbf{D} \to \mathbf{C}$ is a functor and $\gamma : E_C \circ G \to E_D$ is a natural transformation. Suppose there is an adjunction with F left adjoint to G, with unit η and counit ϵ.

If, for any $C \in \mathbf{C}, D \in \mathbf{D}$,

$$1_{E_C(C)} = \phi_C \circ \gamma_{F(C)} \circ E_C(\eta_C) \qquad \text{and} \qquad E_D(\epsilon_D) = \gamma_D \circ \phi_{G(D)}, \tag{1}$$

then the functors $\mathcal{L}(F, \phi) : \mathcal{L}(\mathbf{C}) \to \mathcal{L}(\mathbf{D})$ and $\mathcal{L}(G, \gamma) : \mathcal{L}(\mathbf{D}) \to \mathcal{L}(\mathbf{C})$ form a fibrewise adjunction with $\mathcal{L}(F, \phi)$ left adjoint to $\mathcal{L}(G, \gamma)$ and unit and counit given as follows: the unit at $(C, l : E_C(C) \to L)$ is $(\eta_C, 1_L)$; the counit at $(D, l : E_D(D) \to L)$ is $(\epsilon_D, 1_L)$. If, in addition, the adjunction between F and G is a coreflection or reflection, then $\mathcal{L}(F, \phi)$ and $\mathcal{L}(G, \gamma)$ form a coreflection or reflection respectively.

Proof: By [50] theorem 2 p.81, the adjunction between $\mathbf{C}$ and $\mathbf{D}$, is determined by the functors F, G, the natural transformations η, ϵ and the fact that the compositions

$$G(D) \xrightarrow{\eta_{G(D)}} GFG(D) \xrightarrow{G(\epsilon_D)} G(D),$$
$$F(C) \xrightarrow{F(\eta_C)} FGF(C) \xrightarrow{\epsilon_{F(C)}} F(C)$$

are identities. The condition (1) is sufficient to ensure that these facts lift straightforwardly to the labelled categories and functors, determining an adjunction with unit and counit as claimed. The unit and counit are vertical, making the adjunction fibrewise. Given their form, they become natural isomorphisms if η or ϵ are; the property of being a coreflection or reflection is preserved by the construction.$\square$

This lemma enables us to transport the adjunctions that exist between categories of unlabelled structures to adjunctions between the corresponding categories labelled structures. The role of the natural transformations is to relate the event sets of the image of a functor to the event set of the original object. We are only required to check that the natural transformations, tracking the functors in the labelling category of sets, relate well to the unit and counit, in the sense of (1) above.

As an example we consider how to extend the coreflection between event structures and trace languages to labelled versions of these structures using lemma 89. The role of the natural transformations in the lemma is to relate the event sets of the image of a functor to the event set of the original object, as can be seen by considering the functor

$$tle : \mathbf{TL} \to \mathbf{E}.$$

Let $E_{TL} : \mathbf{TL} \to \mathbf{Set}_*$ be the forgetful functor from trace languages to their alphabets. Let $E_E : \mathbf{E} \to \mathbf{Set}_*$ be the forgetful functor from event structures to their sets of events. A component of the counit of the coreflection between $\mathbf{E}$ and $\mathbf{TL}$ maps the events of a trace language to its alphabet. It yields a natural transformation $\gamma : E_E \circ tle \to E_{TL}$. A trace language $T = (M, A, I)$ with labelling $l : A \to L$ can now be sent to the event structure $tle(T)$ with labelling $l \circ \gamma_T : E \to L$. This extends to a functor $\mathcal{L}(tle, \gamma) : \mathcal{L}(\mathbf{TL}) \to \mathcal{L}(\mathbf{E})$. The functor $etl : \mathbf{E} \to \mathbf{TL}$ does not change the set of events and we associate it with the identity natural transformation $1 : E_{TL} \circ etl \to E_E$. These choices of natural transformations to associate with the functors etl and tle ensure that condition (1) of lemma 89 hold. To see this, we use the fact that

$$\epsilon_{etl(E)} \circ etl(\eta_E) = 1_{etl(E)}$$

obtains for counit ϵ and unit η of the adjunction, for any $E \in \mathbf{E}$. Thus applying the functor E_{TL}, we get

$$E_{TL}(\epsilon_{etl(E)}) \circ E_{TL}(etl(\eta_E)) = E_{TL}(1_{etl(E)}).$$

But now, recalling how E_{TL} and etl act, we see

$$\epsilon_{etl(E)} \circ \eta_E = 1_E,$$

i.e. that the first half of (1) holds. The remaining half of (1) reduces to an obvious equality. We conclude by lemma 89 that

$$\mathcal{L}(etl, 1) : \mathcal{L}(\mathbf{E}) \to \mathcal{L}(\mathbf{TL})$$

forms a coreflection, with right adjoint $\mathcal{L}(tle, \gamma)$. The coreflection $\mathbf{E} \hookrightarrow \mathbf{TL}$ cuts down to one $\mathbf{E} \hookrightarrow \mathbf{TL}_0$, which extends to labelled structures.

So, in particular, we can lift the coreflection between event structures and trace languages to labelled versions of these structures. In a similar, but much easier manner, we can lift the adjunction between $\mathbf{A}$ and $\mathbf{N}$, and the coreflections

$$\mathbf{E} \hookrightarrow \mathbf{TL}_0 \hookrightarrow \mathbf{A}, \qquad \mathbf{E} \hookrightarrow \mathbf{A}_0 \hookrightarrow \mathbf{N}, \qquad \mathbf{A}_c \hookrightarrow \mathbf{Safe},$$

to the categories of labelled structures. Lemma 89 requires that each functor is associated with a natural transformation relating the events of the image to those originally. In most cases the functors leave the event sets unchanged, which makes the identity natural transformations the evident associates of the adjoint functors and the verification of condition (1) of lemma 89 a triviality. One exception is right adjoint of the coreflection from $\mathbf{TL}_0$ to $\mathbf{E}$, dealt with in section 11.2. Another is the functor $na_0 : \mathbf{N} \to \mathbf{A}_0$ which has the effect on event sets of reducing them to those events which are reachable. Accordingly, when extending this functor to labelled structures we take the natural transformation associated with na_0 to have components the inclusion of the events of $na_0(N)$ in those of a net N. A straightforward application of lemma 89 lifts the coreflection between asynchronous transition systems and nets to labelled structures. We obtain an adjunction between $\mathcal{L}(\mathbf{A})$ and $\mathcal{L}(\mathbf{N})$, and the coreflections

$$\mathcal{L}(\mathbf{E}) \hookrightarrow \mathcal{L}(\mathbf{TL}_0) \hookrightarrow \mathcal{L}(\mathbf{A}), \quad \mathcal{L}(\mathbf{E}) \hookrightarrow \mathcal{L}(\mathbf{A}_0) \hookrightarrow \mathcal{L}(\mathbf{N}), \quad \mathcal{L}(\mathbf{A}_c) \hookrightarrow \mathcal{L}(\mathbf{Safe}).$$

We can use the adjunctions to relate constructions, and thus semantics, across different categories of labelled structures.

In section 8.3.3 we saw that the reflection between languages and synchronisation trees is paralleled by a reflection between Mazurkiewicz trace languages and labelled event structures. Several independence models are generalisations of transition systems: labelled Petri nets, labelled asynchronous transition systems, transition systems with independence. There is a coreflection $\mathbf{T} \hookrightarrow \mathbf{TI}$ given by regarding a transition system as having empty independence relation. However there are not coreflections from transition systems $\mathbf{T}$ to the categories of labelled nets or asynchronous transition systems. There are not for the irritating reason that, unlike transition systems, these two models allow more than one transition with the same label between two states. This stops the natural bijection required for the "inclusion" of transition systems to be a left adjoint. A more detailed comparison between interleaving and independence models can suggest new models. See, for example, [81, 82] for a classification of models which includes a generalisation of Mazurkiewicz traces, associated with a broad class of pomset languages.

Remark: An alternative scheme of labelling is possible for the independence models. Instead of labelling events simply by *sets* of labels, in $\mathbf{Set}_*$, we can label by sets together with an independence relation, in the category $\mathbf{Set}_I$, respecting the independence on events in the labelling function. We met the category $\mathbf{Set}_I$ of sets with independence earlier in section 8.3.3 and, as an example of the general construction, the category $\mathcal{L}_I(\mathbf{E})$ of event structures labelled by sets with independence. The general construction for labelling in $\mathbf{Set}_I$ proceeds as with $\mathbf{Set}_*$ and similar general lemmas apply; in particular, the adjunctions/coreflections/reflections between the categories of unlabelled structures lift when labelling by sets with independence. For all the independence models but nets the evident projection functors from the categories with labelling sets with independence to $\mathbf{Set}_I$ are bifibrations—for nets we get just cofibrations. This contrasts with the situation when labelling the independence models by $\mathbf{Set}_*$. Then the associated projections only form cofibrations, not fibrations; while the labelled categories do have cartesian liftings of total maps between labelling sets, they do not have cartesian liftings of truly partial maps. A limitation with labelling by sets with independence is that the relabelling construction on processes is determined as a cocartesian lifting only when the relabelling function is a morphism of the category $\mathbf{Set}_I$, and so preserves independence. The categories labelling by plain sets are more suitable for the semantics of traditional process algebras. Still, we can imagine process algebras, in which processes have sorts consisting of sets with independence, with relabelling operations only for relabelling functions preserving independence.

13 Notes

In this chapter we have surveyed a number of models for concurrency, with special emphasis on the use of category theory in relating the models. We have chosen not to go into any detail on the theories and applications of the individual models. In the following we give some references to such work, and work related to our presentation in general.

Labelled transition systems are arguably the most fundamental model within theoretical computer science. An early reference is Keller [42]. In the context of concurrency, they are central to the work on process algebras, like CCS, where processes are typically first given a semantics as labelled transition systems on which behavioural equivalences, like bisimulation, and logics, like Hennessy-Milner logic, are then defined. For one prototypical example of such an approach, see Milner's treatment of CCS in [55], and for surveys of the many equivalences and logics which have been studied see the papers of van Glabbeek [26] and [27].

Labelled transition systems have been introduced here in their most basic form. Many extensions have been suggested and studied. As a simple example, we have assumed each transition system has one and only one initial state. A similar theory can be developed with a set of initial states, the interpretation being that initially one and only one of the initial states holds, though it is not determined which (a notable difference is that then the coproduct amounts to just disjoint juxtaposition). More significant extensions include explicit representations of concepts like fair, timed and probabilistic behavours. Using labelled transition systems as a model for distributed systems, one often needs to restrict the set of infinite behaviours to those which meet certain progress assumptions for the individual components of the system. Extensions of labelled transition systems dealing with this and other notions of fairness may be found in the works of Manna and Pnueli [51]. Recently, a lot of attention has been paid to extensions taking an explicit account of timing aspects, *e.g.* associating a time measure to each transition, see *e.g.* [63]. Some work has also been done on versions of labelled transition systems extended with probability distributions associated with non-deterministic branching, as in [47]. For all three types of extensions, generalised theories of equivalences and logics have been developed. Specifications typically take the form of an existing formalism extended to fair, timed or probabilistic behaviours.

Synchronisation trees appeared in Milner's early work of CCS [54]. We use the term in a more general sense, of trees in which arcs are labelled by actions which may be, but are not exclusively, CCS actions.

It is fairly common to see languages, or sets of sequences of states, used to give semantics to parallel processes. The expression *Hoare traces* often turns up in this context, stemming from Hoare's article [33] though the idea did not originate there, for example appearing in the early work on path expressions [48].

Hoare traces and synchronisation trees represent two extremes in a variety of views on the branching structure of behaviours, often referred to as the linear time versus branching time spectrum. In the literature they have been given names like acceptance, refusal, ready, and failure semantics. For a comprehensive survey of these views in terms of models, equivalences and logics see [26, 27]. The models in between Hoare traces and synchronisation trees are typically defined in terms of languages of strings decorated with some branching information. For a thorough treatment of one such model (acceptance trees and testing) see Hennessy's book [35].

The "partial simulation" morphisms we define on transition systems seem to have been discovered several times. They bear a close relationship to bisimulation, as pointed out by several authors, *e.g.* [39]. Their relevance to the operators of proces algebras like CCS and CSP was first pointed out by Winskel in [94]. Because morphisms relate the behaviour of a constructed transition system to that of its components, they also play a role in compositional reasoning (see *e.g.* [100]).

One omission from our categorical explication of operators is a treatment of hiding, in which certain specified actions are made internal. In the case of languages, such an operation of hiding is achieved by the functor $\lambda_!$ associated with cocartesian lifting; even when λ is partial, and taken to be undefined on the actions to be hidden, it has cocartesian liftings. But this operation does not seem to capture hiding correctly on the branching structures of transition systems and synchronisation trees. Prefixing might also be expected to play a deeper role categorically than it does at present.

Synchronisation algebras were used largely for the purpose of generality in [92]. They can be regarded as generalising Milner's monoids of actions [56] by allowing asynchrony between processes (however, here we are on sticky ground, as Milner's monoids are open to different interpretations). A similar idea appeared independently in the work of Aczel, and also Bergstra and Klop [7].

Equivalences between operational and denotational semantics, like the one presented for our process language in section 3, are well known from sequential programming languages, *e.g.* [101]. Here the operational semantics is given in a syntax-directed way using Plotkin's structural operational semantics, SOS [74], and the denotational semantics based on the complete partial order approach of Scott [83]. Plotkin's SOS approach has also been used to give operational semantics for high level process languages with value-passing like OCCAM (see *e.g.* [18]). Many equivalence results between operational and denotational semantics exist for the linear time versus branching time spectrum mentioned above, see *e.g.* [15, 16, 35]).

An early reference for Mazurkiewicz traces is [61], though the material can also be found in [62]. Mazurkiewicz traces are generally defined a little differently. In particular it is not usual to insist on the *coherence* axiom (referred to as *properness* by Mazurkiewicz [62]) in their definition.

As remarked, Mazurkiewicz traces may be viewed as special kinds of labelled partial orders of events. Labelled partial orders of events appeared earlier in the study of concurrency by Lamport [46] and Petri [71], and has been advocated under the name of pomsets by Pratt in a series of papers beginning with [76], and by Grabowski under the name of partial words [30]. Note that far from all pomsets can be seen as Mazurkiewicz traces. Consequently Mazurkiewicz trace languages correspond to special kinds of pomset languages (see [30], [9] and [82] for some results on their formal relationship). Temporal logics for partially ordered behaviours have been studied by Pinter and Wolper [73] and Katz and Peled [41].

On the other hand, Mazurkiewicz traces may also be seen as a generalisation of normal strings (with the extra notion of independence between letters), and, following this view, much of the theory of classical formal language theory has been lifted to trace languages. As an example, regular trace languages have been characterized by acceptors (the asynchronous automata of Zielonka [103]), algebraically (by Ochmanski [66]) and logically (by Thomas [89]).

Event structures of the kind treated here were introduced by Plotkin and the authors in [64], and the theory of these and generalized event structures developed by Winskel [91, 95, 97, 98]. Event structures can have a general, and not just a binary conflict, and so can represent precisely the dI-domains of Berry (not just the coherent ones). Through replacing the partial order of causal dependency by an enabling relation, they can represent nondistributive domains. Stable event structures bear the same relation to dI-domains as do information systems to Scott domains. The article [97] gives a reasonable survey, and see [24] for some extensions. The characterisations of the domains of configurations as prime algebraic appear in [64] and [91], and the realisation that prime algebraicity amounts to precisely distributivity in [98, 93].

The difficulty in defining operations like products and parallel compositions on event structures of the form $(E, \leq, \#)$ has encouraged the use of more general event structures with which it is easier to give semantics to parallel programming languages, or even languages with higher types (see [97, 98]). Provided the more general event structures have coherent dI-domains as domains of configurations an event structure of the form $(E, \leq, \#)$ can always be extracted as its complete primes. This line has been followed in [92, 97, 11]. The method is similar to that of using another model like trace languages, asynchronous transition systems or Petri nets to give a semantics, from which an event-structure semantics is then induced by the coreflection. Event-structure semantics for CCS/TCSP-like languages was made systematic by Winskel in [92], which exploited a new definition of morphism—that which appears here.

A variation of event structures as models for process languages appear in the "flow event structures" of Boudol and Castellani developed in [11, 13]; here the problems with the definition of parallel composition is overcome at the expense of an unusual treatment of restriction, one where the events to be restricted away are made self-conflicting instead of removed [20]. Other variations include the bundle event structures of Langerak [49], the families of posets of Rensink [79], and the event automata of Gunawardena [32] and Pinna and Poigne [72].

Like most of our models, event structures have been equipped with notions of behavioural equivalences (like the history preserving bisimlation of Rabinovich and Trakhtenbrot [78]) and logics (for some axiomatizations see the works of Mukund, Thiagarajan [58, 59] and Penczek [70]).

The relationship between event structures and Mazurkiewicz trace languages seems first to have been made explicit by Bednarczyk [5]. However, the proof of the representation theorem here appears to be new. For an alternative proof see Rozoy and Thiagarajan [80].

A good reference on *Petri nets* is [2]. The version of Petri nets we describe can be found in the paper [62] of Mazurkiewicz. They are more general than Thiagarajan's elementary net systems [88] because they allow an event to occur even when there is a condition which is simultaneously a pre and post condition. There is a well-known technique known as "complementation" for making a non-safe net safe. It is notable that this construction comes out of the adjunction between nets and asynchronous transition systems. Our version of Petri nets has been used as semantic model for process languages in the works of *e.g.* Olderog [68].

There are several versions of morphisms on nets in the literature, some more deserving of attention than others. We have examined two. The original definition by Petri [71] seems to have been motivated by graph-theoretic considerations—Petri's morphisms do not respect the behaviour of nets. To some extent the ideas presented here generalise to nets in which events can fire and markings hold with multiplicities, as indicated in [99], though at present it is not known how to link up with other models via adjunctions. See also Meseguer and Montanari's study of several definitions of net morphisms [53].

Categories of Petri nets have been shown to form a model of Girard's linear logic, offering an interpretation of the logical operations of linear logic as operations on nets and of proofs as kinds of simulation morphisms like those here (see [17]). Since the morphisms preserve behaviour, the existence of a morphism from one net to another may be interpreted as saying that one net (the implementation) satisfies another (the specification). Recently categories of games have been shown to be models of linear classical logic [1, 36, 45, 21]. The games have the structure of special Petri nets in which the distinction between moves of a player and opponent is made through one being conditions and the other events (linear negation is caught as reversal of the roles of the players corresponding to swapping the nature of conditions and events). Morphisms are identified with (partial) strategies. As well as providing a refinement of Berry and Curien's sequential algorithms the new categories are suggestive of new paradigms for computation.

Asynchronous transition systems are due to Bednarczyk [5] and Shields [84] who discovered them independently. Bednarczyk's thesis [5] contains the definition of the category of asynchronous transition systems and the coreflections with event structures and Mazurkiewicz traces. Transition systems with independence are related to the concurrent transition systems of Stark [85]. A related "geometric" approach towards noninterleaving transition systems is taken by Pratt in [77] and Goubault and Jensen in [29].

As remarked, when presented as transition systems with independence, asynchronous transition systems are amenable to the same techniques (*e.g.* definition by structural operational semantics) as ordinary transition systems. Alternatively, asynchronous transition systems can arise directly through operational semantics, but where instead of just labels, transitions carry more complicated information from which event names and independence can be extracted (see [12, 60, 3] for three examples of this approach). The use of asynchronous transition systems in semantics is often less clumsy than that of nets, which can be extracted afterwards via the adjunction with nets—though sometimes care must be taken to show that the constructions used stay within $\mathbf{A}_0$.

The adjunction between asynchronous transition systems and nets is new. It can be viewed as an extension of the adjunction between elementary net systems and elementary transition systems of Nielsen, Rozenberg and Thiagarajan [65]. A similar result for general place transition nets and a class of transition systems is presented by Mukund in [57].

A lot of attention has been paid to noninterleaving semantics in the presence of operators changing the level of atomicity of actions. It turns out that such operators do not admit compositional definitions in the interleaving models, and hence they motivate directly the study of noninterleaving. For examples, see the complementary works of Vogler [90] and Boudol [10]. Other examples of equivalences of operationlly and denotationally defined noninterleaving semantics have been provided by Boudol and Castellani [13], Degano, De Nicola and Montanari [23], Gorrieri [28], and Mukund and Nielsen [60].

Transition systems play an important role in model checking. Interestingly, the extra notion of independence has recently proved to be of value in the search for efficient model checkers—see the works of Wolper [102].

We have chosen in our treatment not to discuss the various equivalences or logics that one might impose on the models. A good survey of the intimidating range of possible equivalences is given in [25] and for noninterleaving models see *e.g.* [22]. Some evidence that categorical ideas might help clean up the mess, is contained in [39]; there a method is shown for obtaining a generalisation of bisimulation equivalence on categories of models like those here. A central notion in [39] is that of *open map* which, restored to the topos setting where it originated, yields presheaf categories as models, into which synchronisation trees and labelled event structures embed fully and faithfully.

The work on relating models for concurrency has been pursued by others, like Bednarczyk [5], Rensink [79], and Kwiatkowska [43]. A fuller picture than the one presented here in section 12 has been worked out by Sassone and the authors [81]. For good introductions to category theory we refer to [4] and [50].

Haunting this survey of models for concurrency and their relationship has been the feeling, from time to time, that perhaps the existing models are not quite the right ones, that the lack of existence of certain operations is due to an inadequacy in the models as they are traditionally presented. As our knowledge and experience of what is required of languages and models for parallel computation increases, we will surely be led to richer models and to understand better what structure they should possess. And whatever their form, we should understand how the new models fit with the traditional models studied here.

Acknowledgements

We are grateful to P.S.Thiagarajan, Bart Jacobs, Nils Klarlund, Peter Knijnenburg, Vladimiro Sassone and Rob van Glabeek for helpful suggestions. In particular, Bart Jacobs supplied the short proofs for the appendix on fibred categories. Allan Cheng and Bettina Blaaberg Sørensen are to be thanked for spotting several errors in an earlier draft. Thanks to Uffe Engberg and Madhaven Mukund for their preparation of several of the diagrams.

A A basic category

We shall work with a particular representation of the category of sets with partial functions. Assume that X and Y are sets not containing the distinguished symbol $*$. Write $f : X \to_* Y$ for a function $f : X \cup \{*\} \to Y \cup \{*\}$ such that $f(*) = *$. When $f(x) = *$, for $x \in X$, we say $f(x)$ is *undefined* and otherwise *defined*. We say $f : X \to_* Y$ is *total* when $f(x)$ is defined for all $x \in X$. Of course, such total morphisms $X \to_* Y$ correspond to the usual total functions $X \to Y$, with which they shall be identified. For the category $\mathbf{Set}_*$, we take as objects sets which do not contain $*$, and as morphisms functions $f : X \to_* Y$, with the composition of two such functions being the usual composition of total functions (but on sets extended by $*$). Of course, $\mathbf{Set}_*$ is isomorphic to the category of sets with partial functions, as usually presented.

We remark on some categorical constructions in $\mathbf{Set}_*$. A coproduct of X and Y in $\mathbf{Set}_*$ is the disjoint union $X \uplus Y$ with the obvious injections. A product of X and Y in $\mathbf{Set}_*$ has the form $X \times_* Y =$

$$\{(x, *) \mid x \in X\} \cup \{(*, y) \mid y \in Y\} \cup \{(x, y) \mid x \in X, y \in Y\}$$

with projections those partial functions to the left and right coordinates.

B Fibred categories

Our presentation relies on some basic notions from fibred category theory originating in the work of Grothendieck [31], and Bénabou [6].[12]

Definition: Let $p : \mathbf{X} \to \mathbf{B}$ be a functor.

A morphism $f : X \to X'$ in $\mathbf{X}$ is said to be *cartesian* with respect to p if for any morphism $g : Y \to X'$ in $\mathbf{X}$ such that $p(g) = p(f)$ there is a unique morphism $h : Y \to X$ such that $p(h) = 1_{p(X)}$ and $f \circ h = g$. As a diagram:

$$\begin{tikzcd}
 & Y \arrow[d, dashed, "h"'] \arrow[dr, "g"] & \\
\mathbf{X} \arrow[dd, "p"'] & X \arrow[r, "f"] & X' \\
 & & \\
\mathbf{B} & p(X) \arrow[r, "p(f)"] & p(X')
\end{tikzcd}$$

A cartesian morphism $f : X \to X'$ in $\mathbf{X}$ is said to be a *cartesian lifting* of the morphism $p(f)$ in $\mathbf{B}$ with respect to X'.
Say $p : \mathbf{X} \to \mathbf{B}$ is a *fibration* if

[12] Our presentation has been improved by incorporating proofs of Bart Jacobs.

- every morphism $\lambda : B \to B'$ in $\mathbf{B}$ has a cartesian lifting with respect to any X' such that $p(X') = B'$, and
- any composition of cartesian morphisms is again cartesian.

A morphism $f : X \to X'$ in $\mathbf{X}$ is said to be *vertical* if $p(f) = 1_{p(X)}$.

Often p is called the *projection*, $\mathbf{B}$ the *base* category, and each subcategory $p^{-1}(B)$ of $\mathbf{X}$, which is sent to the subcategory consisting of the identity morphism on an object B of $\mathbf{B}$, the *fibre* over B.

A fibration can also be presented a little differently. A morphism $f : X \to X'$ in $\mathbf{X}$ is said to be *strong cartesian* with respect to a functor $p : \mathbf{X} \to \mathbf{B}$ if for any $g : Y \to X'$ in $\mathbf{X}$ and morphism $\lambda : p(Y) \to p(X)$ in $\mathbf{B}$ for which $p(f) \circ \lambda = p(g)$ there is a unique morphism $h : Y \to X$ such that $p(h) = \lambda$ and $f \circ h = g$. It is not hard to show that strong cartesian morphisms compose and that any strong cartesian morphism is cartesian. Moreover in a fibration any cartesian morphism is strong cartesian (again not hard to show). Hence a fibration can alternatively be defined as a functor $p : \mathbf{X} \to \mathbf{B}$ for which each morphism in the base category possesses a strong cartesian lifting (without needing the further requirement that cartesian maps compose).

Definition: Let $p : \mathbf{X} \to \mathbf{B}$ be a functor. It is a *cofibration* if $p^{op} : \mathbf{X}^{op} \to \mathbf{B}^{op}$ is a fibration. A morphism $f : X \to X'$ in $\mathbf{X}$ is said to be *cocartesian* with respect to p if f^{op} is cartesian in the fibration; the morphism f is a *cocartesian lifting* of $p(f)$. (Call f strong cocartesian if f^{op} is strong cartesian.) We say p is a *bifibration* if it is both a fibration and cofibration.

Thus a morphism $f : X \to X'$ in $\mathbf{X}$ is cocartesian with respect to p if for any morphism $g : X \to Y$ in $\mathbf{X}$ such that $p(g) = p(f)$ there is a unique morphism $h : X' \to Y$ such that $p(h) = 1_{p(X')}$ and $h \circ f = g$. As a diagram:

$$
\begin{tikzcd}
 & & Y' \\
X \arrow[dd, "p"'] & X \arrow[ur, "g"] \arrow[r, "f"] & X' \arrow[u, dashed, "h"'] \\
 & & \\
\mathbf{B} & p(X) \arrow[r, "p(f)"] & p(X')
\end{tikzcd}
$$

For later proofs it is convenient to have a characterisation of (strong) cartesian morphisms. Let $p : \mathbf{X} \to \mathbf{B}$ be a functor. For X, X' in $\mathbf{X}$ and $\lambda : p(X) \to p(X')$ in $\mathbf{B}$, write

$$\mathbf{X}_\lambda(X, X') =_{def} \{f : X \to X' \mid p(f) = \lambda\}.$$

It is easily verified that:

Proposition 90
A morphism $f : X \to X'$ in $\mathbf{X}$, with $p(f) = \lambda$, is strong cartesian iff for each X'' in $\mathbf{X}$ and $\lambda' : p(X'') \to p(X)$, the map

$$(f \circ -) : \mathbf{X}_{\lambda'}(X'', X) \to X_{\lambda \circ \lambda''}(X'', X'),$$

obtained by composing with f, is an isomorphism (of sets, i.e. a bijection).
A morphism $f : X \to X'$ in $\mathbf{X}$, with $p(f) = \lambda$, is cartesian iff for each X'' in $\mathbf{X}$, the map

$$(f \circ -) : \mathbf{X}_{1_{p(X)}}(X'', X) \to X_{\lambda \circ \lambda''}(X'', X'),$$

obtained by composing with f, is an isomorphism.

(Strong) cocartesian morphisms can be characterised similarly.

We are also concerned with functors $F : \mathbf{X} \to \mathbf{Y}$ between fibrations $p : \mathbf{X} \to \mathbf{B}$ and $q : \mathbf{Y} \to \mathbf{B}$. The functors will preserve the base category in the sense that

$$q \circ F = p.$$

Such functors are said to be *cartesian* when they preserve cartesian morphisms. As the next lemma shows, this property will be automatic for right adjoints of *fibrewise* adjunctions, *i.e.* those in which the functors preserve the base category and cut down to adjunctions between fibres over common objects in the base category. A dual result holds for left adjoints and cofibrations.

Definition: Suppose $p : \mathbf{X} \to \mathbf{B}$ and $q : \mathbf{Y} \to \mathbf{B}$ and that functors $F : \mathbf{X} \to \mathbf{Y}$ and $G : \mathbf{Y} \to \mathbf{X}$ form an adjunction with F left adjoint to G. The adjunction is said to be *fibrewise*, with respect to p and q, iff $q \circ F = p$ and $p \circ G = q$ and each component of the counit $\epsilon_Y : FG(Y) \to Y$ is vertical, for $Y \in \mathbf{Y}$, *i.e.* $q(\epsilon_Y) = 1_{q(Y)}$ (or equivalently, components of the unit are vertical).

Lemma 91 *Suppose $p : \mathbf{X} \to \mathbf{B}$ and $q : \mathbf{Y} \to \mathbf{B}$ and that functors $F : \mathbf{X} \to \mathbf{Y}$ and $G : \mathbf{Y} \to \mathbf{X}$ form a fibrewise adjunction with F left adjoint to G. Then the right adjoint G preserves strong cartesian morphisms, and cartesian morphisms.*

Proof: Assume $g : Y \to Y'$ in $\mathbf{Y}$ is strong cartesian over β in $\mathbf{B}$.

Naturality of the adjunction yields that the diagram

$$\begin{tikzcd}
\mathbf{X}(X, G(Y)) \arrow[r, phantom, "\overset{\theta}{\cong}"] \arrow[d, "(G(g)\circ -)"'] & \mathbf{Y}(F(X), Y) \arrow[d, "(g\circ -)"] \\
\mathbf{X}(X, G(Y')) \arrow[r, phantom, "\overset{\theta'}{\cong}"] & \mathbf{Y}(F(X), Y')
\end{tikzcd}$$

commutes, where $X \in \mathbf{X}$ and θ, θ' are components of the natural isomorphisms of the adjunction. Because the adjunction is fibrewise, $p(-) = q \circ \theta(-)$.

Letting $X \in \mathbf{X}$ and $\alpha : p(X) \to pG(Y)$, we observe the following chain of isomorphisms

$$\begin{array}{lcl}
\mathbf{X}_\alpha(X, G(Y)) & \stackrel{\theta}{\cong} & \mathbf{Y}_\alpha(F(X), Y) \\
 & \stackrel{(g \circ -)}{\cong} & \mathbf{Y}_{\beta\circ\alpha}(F(X), Y') \text{ as } g \text{ is strong cartesian} \\
 & \stackrel{\theta'^{-1}}{\cong} & \mathbf{X}_{\beta\circ\alpha}(X, G(Y'))
\end{array}$$

whose composition, $\theta'^{-1}(g \circ \theta(-))$, equals $(G(g) \circ -)$ from the commuting diagram above. Hence $G(g)$ is strong cartesian by proposition 90.

The proof in the case of just cartesian morphisms is got by specialising the morphism α above to the identity. □

Note that lemma 91 does not state that the left adjoint F preserves cartesian morphisms. Nor does it entail that the right adjoint G preserves cocartesian morphisms, and these are not true in general. For instance, they do not hold for the coreflection between synchronisation trees and transition systems.

In the case where the adjunctions form coreflections (or reflections) there are further useful results. The left adjoint of a coreflection (recall this is an adjunction in which the unit is a natural isomorphism) is automatically full and faithful, as is the right adjoint of a reflection, and thus the following lemma applies to these functors in a fibrewise adjunction:

Lemma 92 *Suppose $p : \mathbf{X} \to \mathbf{B}$ and $q : \mathbf{Y} \to \mathbf{B}$ and that functor $F : \mathbf{X} \to \mathbf{Y}$ is full and faithful with $q \circ F = p$. Then F reflects (stong) (co) cartesian morphisms.*

Proof: The conditions on F imply that F restricts to an isomorphism

$$\mathbf{X}_\gamma(X, X') \stackrel{F}{\cong} \mathbf{Y}_\gamma(F(X), F(X'))$$

for any $X, X' \in \mathbf{X}$ and $\gamma : p(X) \to p(X')$ in $\mathbf{B}$.

Let $X'' \in \mathbf{X}$ and $\alpha : p(X'') \to p(X)$ in $\mathbf{B}$. The chain of isomorphisms

$$\begin{array}{lcl}
\mathbf{X}_\alpha(X'', X) & \stackrel{F}{\cong} & \mathbf{Y}_\alpha(F(X''), F(X)) \\
 & \stackrel{(F(f) \circ -)}{\cong} & \mathbf{Y}_{\beta\circ\alpha}(F(X''), F(X')) \quad \text{as } F(f) \text{ is strong cartesian,} \\
 & \stackrel{F^{-1}}{\cong} & \mathbf{X}_{\beta\circ\alpha}(X'', X')
\end{array}$$

composes to $F^{-1}(F(f) \circ F(-)) = (f \circ -)$. Hence f is strong cartesian by proposition 90.

The proof for just cartesian morphisms follows by specialising α to the identity. The proof for cocartesian liftings is similar. □

C Operational semantics—proofs

Here we provide the proofs required in showing the equivalence between the denotational and operational semantics of the process language in terms of transition systems of section 3.2 (part (a)) and, by a slightly more general argument, section 11.3 (part (b)). Both parts rely on acyclicity of the transition relation got via the operational semantics.

Lemma 10 For any closed tagged term t, the transition system $\mathcal{O}p(t)$ is acyclic.
Proof: We show this by mapping tagged terms t to $| t |$ in a strict order $<$ (an irreflexive, transitive relation) in such a way that

$$t \xrightarrow{a} u \ \& \ a \not\equiv * \Rightarrow | t | < | u | . \tag{1}$$

It then follows that $\rightarrow^+$ is irreflexive.

Define $< \subseteq \omega \times \omega$ by taking $(m, n) < (m', n') \Leftrightarrow \quad m < m'$ or $(m = m' \ \& \ n > n')$. (In other words $<$ is the lexicographic combination of $<$ and $>$ on integers.) The relation $<$ is a strict order. For t a closed tagged term, define

$$| \ t \ | = (tag(t), size(t))$$

where the functions tag and size are defined by the following structural inductions:

$$\begin{array}{lcllcl}
tag(nil) & = & tag(x) = 0 & size(nil) & = & size(x) = 0 \\
tag(at) & = & tag(t) & size(at) & = & 1 + size(t) \\
tag(t_0 \oplus t_1) & = & min(tag(t_0), tag(t_1)) & size(t_0 \oplus t_1) & = & 1 + size(t_0) + size(t_1) \\
tag(t_0 \times t_1) & = & tag(t_0) + tag(t_1) & size(t_0 \times t_1) & = & 1 + size(t_0) + size(t_1) \\
tag(t \restriction \Lambda) & = & tag(t\{\Xi\}) = tag(t) & size(t \restriction \Lambda) & = & size(t\{\Xi\}) = 1 + size(t) \\
tag(rec\, x.t) & = & tag(t) & size(rec\, x.t) & = & 1 + size(t) \\
tag((l, t)) & = & 1 + tag(t). & size((n, t)) & = & 1 + size(t)
\end{array}$$

The rules for operational semantics can be shown to preserve property (1) above, which hence holds of all derivable transitions. For example, considering the rule for recursion, assume

$$| \ t[rec\, x.t/x] \ | < | \ t' \ | .$$

holds of the transition $t[rec\, x.t/x] \xrightarrow{a} t'$ in its premise if $a \neq *$. It follows that

$$tag(t[rec\, x.t/x] \leq tag(t').$$

Clearly $tag(t) \leq tag(t[rec\, x.t/x])$ so

$$tag(rec\, x.t) = tag(t) < \ 1 + \ tag(t') = tag((2, t')).$$

Hence

$$| \ rec\, x.t \ | < | \ (2, t') \ | .$$

holds of the transition $rec\, x.t \xrightarrow{a} (2, t')$. $\square$

(a). Uniqueness for guarded recursions in T

We prove lemma 9, showing that solutions to guarded recursions are unique to within isomorphism. The proof rests on the definition of a family of functors $(-)^{(k)}$, for $k \in \omega$, "projecting" a transition system to the transition system consisting of that part reachable within k steps.

Lemma 93 *Suppose $T_{n,m}$ are transition systems for $n, m \in \omega$ with the property that*

$$T_{n,m} \trianglelefteq T_{n',m'} \text{ when } n \leq n' \text{ and } m \leq m'.$$

Then the set $\{T_{n,m} \mid n, m \in \omega\}$ has a least upper bound

$$\bigcup_{n,m\in\omega} T_{n,m} = \bigcup_{n\in\omega}(\bigcup_{m\in\omega} T_{n,m}) = \bigcup_{m\in\omega}(\bigcup_{n\in\omega} T_{n,m}) = \bigcup_{n\in\omega} T_{n,n}.$$

Proposition 94

1. *For each $k \in \omega$, the operation $(-)^{(k)}$ is a functor on the category* **T** *of transition systems; it restricts to an endofunctor on the subcategory where morphisms are label-preserving.*

2. *Let T be a transition system. Then $T^{(k)} \trianglelefteq T$, for $k \in \omega$. If $k \leq l$ then $T^{(k)} \trianglelefteq T^{(l)}$. For $k, l \in \omega$, $(T^{(k)})^{(l)} = T^{min(k,l)}$. Recalling the operation $\mathcal{R}$ of section 3, taking the reachable part of a transition system, we have*

$$\mathcal{R}(T) = \bigcup_{k\in\omega} T^{(k)}.$$

3. *The operations $(-)^{(k)}$, for $k \in \omega$, and $\mathcal{R}$ are continuous with respect to $\trianglelefteq$.*

Proof: (1) As morphisms preserve or collapse transitions, it follows that a morphism $f : T_0 \to T_1$ restricts to a morphism $f^{(k)} : T_0^{(k)} \to T_1^{(k)}$. The operation $(-)^{(k)}$ clearly preserves identities and composition. It is easily checked that these facts also hold when restricting attention to label-preserving morphisms.

(2) is obvious.

(3) From the definition of $(-)^{(k)}$, for $k \in \omega$, it is easily seen that it is continuous. To show $\mathcal{R}$ is continuous suppose

$$T_0 \trianglelefteq \cdots \trianglelefteq T_n \trianglelefteq \cdots$$

is a chain of transition systems. Then

$$\begin{array}{rcll}
\mathcal{R}(\bigcup_n T_n) & = & \bigcup_k (\bigcup_n T_n)^{(k)} & \text{by the definition of } \mathcal{R} \\
& = & \bigcup_k \bigcup_n T_n^{(k)} & \text{by continuity of } (-)^{(k)} \\
& = & \bigcup_n \bigcup_k T_n^{(k)} & \text{by lemma 93} \\
& = & \bigcup_n \mathcal{R}(T_n) & \text{by the definition of } \mathcal{R}. \square
\end{array}$$

Say an operation F on transition systems is definable if it acts on T so that

$$F(T) = \mathbf{T}[\![t]\!]\rho[T/x]$$

for some choice of process term t and variable x. Any operation F definable in the process language is $\trianglelefteq$-monotonic and continuous and has the property that

$$(F(T))^{(k)} = (F(T^{(k)}))^{(k)}. \tag{1}$$

This follows by structural induction from facts such as

$$(T \times U)^{(k)} = (T^{(k)} \times U^{(k)})^{(k)}$$

about the basic operations. A prefixing operation $a(-)$ has the stronger property that, for $k > 0$,

$$(a(T))^{(k)} = (a(T^{(k-1)}))^{(k)}.$$

It follows that an operation F defined by a guarded recursion satisfies

$$(F(T))^{(k)} = (F(T^{(k-1)}))^{(k)} \tag{2}$$

for $k > 0$. All the operations extend to functors with respect to label-preserving morphisms on transition systems. The facts above extend to morphisms. A functor F defined by a guarded recursion has the property that

$$(F(f))^{(k)} = (F(f^{(k-1)}))^{(k)} \tag{3}$$

for $k > 0$, on label-preserving morphisms f.

Definition: We will call a functor F on the category of transition systems with label-preserving morphisms *guarded* when it satisfies (2) and (3) above.

Now we can complete the proof of lemma 9:
Lemma 9 If F is defined from a guarded recursion we have:

$$T \cong \mathcal{R} \circ F(T) \Rightarrow T \cong \mathcal{R}(fix(F)).$$

(Here, and in the proof below, morphisms are understood to be in the category with label-preserving morphisms. In particular, the isomorphisms above are label-preserving.)

Proof: Any functor, on the category of transition systems with label-preserving morphisms, definable in the process language is $\trianglelefteq$-continuous. In particular, we remark that such a guarded F has the property that

$$(fix(F))^{(n)} = (F^k(I))^{(n)} \tag{4}$$

for all $k, n \in \omega$ for which $k \geq n$. This is obviously so for $n = 0$ when $(fix(F))^0 = I = (F^k(I))^{(0)}$, where I is the transition system consisting of a single initial state. Assume (4) as induction hypothesis. We deduce, assuming $k \geq n+1$, that

$$\begin{array}{rcll}
(F^k(I))^{(n+1)} & = & (F(F^{k-1}(I)))^{(n+1)} & \\
& = & (F((F^{k-1}(I))^{(n)}))^{(n+1)} & \text{as } F \text{ is guarded} \\
& = & (F((fix(F))^{(n)})^{(n+1)} & \text{from the induction hypothesis as } k-1 \geq n \\
& = & (F(fix(F)))^{(n+1)} & \text{as } F \text{ is guarded} \\
& = & (fix(F))^{(n+1)}. &
\end{array}$$

We conclude (4) holds for general $n \in \omega$, with $k \geq n$, by induction.

With the help of an observation we can simplify the proof notationally. For an operation F definable in the process language

$$\mathcal{R} \circ F = \mathcal{R} \circ F \circ \mathcal{R}. \tag{5}$$

To see this, for an arbitrary transition system T, reason that

$$\begin{array}{rcll}
\mathcal{R} \circ F \circ \mathcal{R}(T) & = & \mathcal{R} \circ F(\bigcup_k T^{(k)}) & \text{by definition of } \mathcal{R}, \\
& = & \bigcup_k \mathcal{R} \circ F(T^{(k)}) & \text{by continuity of } \mathcal{R} \text{ and } F, \\
& = & \bigcup_k \bigcup_n (F(T^{(k)}))^{(n)} & \text{by definition of } \mathcal{R}, \\
& = & \bigcup_k (F(T^{(k)}))^{(k)} & \text{by lemma 93} \\
& = & \bigcup_k (F(T))^{(k)} & \text{by (1)} \\
& = & \mathcal{R} \circ F(T) & \text{by definition of } \mathcal{R}
\end{array}$$

It follows that

$$\begin{array}{rcll}
\mathcal{R}(fix(F)) & = & \mathcal{R}(\bigcup_n F^n(I)) & \\
& = & \bigcup_n \mathcal{R}\, F^n(I) & \\
& = & \bigcup_n (\mathcal{R} \circ F)^n \mathcal{R}(I) & \text{by repeated use of (5)} \\
& = & \bigcup_n (\mathcal{R} \circ F)^n (I) & \\
& = & fix(\mathcal{R} \circ F). &
\end{array}$$

Hence, writing $G = \mathcal{R} \circ F$, we can restate the goal of our proof (in the statement of the theorem) as

$$T \cong G(T) \Rightarrow T \cong fix(G) \tag{\dagger}$$

where we have G is $\trianglelefteq$-continuous and guarded (*i.e.* satisfies (2) and (3)), because these properties are assumed of F and inherited by G.

To prove (†), assume $T \cong G(T)$ and let

$$\theta : G(T) \cong T$$

name the isomorphism. It is also convenient to let u be the unique morphism

$$u : I \rightarrow T.$$

By induction on n we show

$$(G^n(u))^{(n)} : (G^n(I))^{(n)} \to (G^n(T))^{(n)}$$

is an isomorphism. The basis case amounts to showing $u^{(0)} : I^{(0)} \to T^{(0)}$ is an isomorphism which is trivially so as each transition system consists only of a single initial state. Notice that

$$\begin{array}{rcl}(G^{n+1}(u))^{(n+1)} & = & (G(G^n(u)))^{(n+1)} \\ & = & (G((G^n(u))^{(n)}))^{(n+1)},\end{array}$$

because G is guarded. From the fact that G and $(-)^{(n+1)}$ are functors and so preserve isomorphism, we now see that $(G^{(n+1)}(u))^{(n+1)}$ being an isomorphism follows inductively from $(G^n(u))^{(n)}$ being an isomorphism.

Let θ_n be the isomorphism

$$(\theta \circ G(\theta) \circ \cdots \circ G^{(n-1)}(\theta)) : G^n(T) \to T$$

for $n \in \omega$. The fact that $(-)^{(n)}$ is a functor ensures $\theta_n^{(n)}$ is also an isomorphism

$$G^n(T)^{(n)} \to T^{(n)},$$

for $n \in \omega$. As remarked in (4) above, $(fix(G))^{(n)} = (G^n(I))^{(n)}$. Thus we obtain isomorphisms

$$\phi_n =_{def} (\theta_n \circ (G^n(u)))^{(n)} : (fix(G))^{(n)} \to T^{(n)}$$

for $n \in \omega$. These are consistent in the sense that

$$\phi_n = \phi_{n+1}^{(n)},$$

for $n \in \omega$, as we will now show.

Let j be the monomorphism associated with $I \trianglelefteq G(I)$. Applying G^n followed by $(-)^{(n)}$, we obtain an inclusion morphism

$$(G^n(j))^{(n)} : (G^n(I))^{(n)} \to (G^{n+1}(I))^{(n)}.$$

But now by (4),

$$(G^n(j))^{(n)} : (fix(G))^{(n)} \to (fix(G))^{(n)}$$

which being an inclusion morphism must be the identity, *i.e.*

$$(G^n(j))^{(n)} = 1_{(fix(G))^{(n)}}.$$

Certainly we have

$$u = \theta \circ G(u) \circ j,$$

which may be depicted by the following commuting diagram:

$$\begin{array}{ccc} I & \stackrel{u}{\to} & T \\ j\downarrow & & \uparrow\theta \\ G(I) & \stackrel{G(u)}{\to} & G(T) \end{array}$$

Hence, for $n \in \omega$, applying the functors G^n and $(-)^{(n)}$, we obtain

$$\begin{aligned}(G^n(u))^{(n)} &= (G^n(\theta))^{(n)} \circ (G^{n+1}(u))^{(n)} \circ (G^n(j))^{(n)} \\ &= (G^n(\theta))^{(n)} \circ (G^{n+1}(u))^{(n)}.\end{aligned}$$

Thus

$$\begin{aligned}\phi_n &= (\theta_n \circ G^n(u))^{(n)} \\ &= (\theta \circ \cdots \circ G^{(n-1)}(\theta))^{(n)} \circ (G^n(\theta))^{(n)} \circ (G^{n+1}(u))^{(n)} \\ &= ((\theta_{n+1} \circ G^{n+1}(u))^{(n+1)})^{(n)} \text{ by proposition 94(2)} \\ &= \phi_{n+1}^{(n)}.\end{aligned}$$

It follows that

$$\phi = \bigcup_{n\in\omega} \phi_n$$

is an isomorphism $fix(G) \to T$. □

(b). Semantics in TI

This part of the Appendix is dedicated to showing the equivalence between the operational and denotational semantics of **Proc** in terms of transition systems with independence:

Theorem 87 Let t be a closed process term. Then, for an arbitrary environment ρ,

$$\mathcal{O}p(t) \cong \mathcal{R}(\mathbf{TI}[\![t]\!]\rho),$$

a label-preserving isomorphism of transition systems with independence.

The proof is very like that of theorem 11. However in carrying out the proof we move to more general structures than transition systems with independence, to pre-structures of the same form but which do not necessarily satisfy the axioms required of a transition system with independence. This is because *a priori* we do not know that $\mathcal{O}p(t)$ is a transition system with independence for t a closed term.

Definition: Define **pTI** to be the category consisting of

- objects $(S, i, L, \mathit{Tran}, I)$ where (S, i, L, Tran) is a transition system and $I \subseteq \mathit{Tran} \times \mathit{Tran}$ (the independence relation),
- morphisms are morphisms between the underlying transition systems which preserve independence, *i.e.* a morphism $(\sigma, \lambda) : T \to T'$ should satisfy

  If (s, a, s') and (u, b, u') are independent transitions of T and $\lambda(a)$ and $\lambda(b)$ are both defined, then $(\sigma(s), \lambda(a), \sigma(s'))$ and $(\sigma(u), \lambda(b), \sigma(u'))$ are independent transitions of T'.

  Composition is inherited from that in $\mathbf{T}$.

It is clear that $\mathbf{TI}$ is a full subcategory of $\mathbf{pTI}$ and the two categories share many constructions. The same definitions of the process language operations, given in section 11.3.1 for $\mathbf{TI}$, serve also for $\mathbf{pTI}$. In particular, the definition of $\trianglelefteq$ can be lifted to give a least-fixed-point semantics of recursion. All told, we can give a denotational semantics of terms in $\mathbf{Proc}$ as objects of $\mathbf{pTI}$; for a term t of $\mathbf{Proc}$, with respect to an environment ρ from process variables to objects of $\mathbf{pTI}$ we obtain a denotation

$$\mathbf{pTI}[\![t]\!]\rho,$$

an object in $\mathbf{pTI}$. When the environment ρ yields objects in the subcategory $\mathbf{TI}$ for any free variable of t, the two denotations, $\mathbf{pTI}[\![t]\!]\rho$ and $\mathbf{TI}[\![t]\!]\rho$ coincide.

As in (a), a key idea is that of an endofunctor $(-)^{(k)}$ on $\mathbf{pTI}$ "projecting" an object to that part of it which is reachable within k steps, taking T to $T^{(k)} \trianglelefteq T$. In detail:

Definition: Let $T = (S, i, L, \mathit{Tran}, I)$ be in $\mathbf{pTI}$. Define $T^{(k)}$ to be $(S', i, L', \mathit{Tran}', I')$ where S' is the subset of states S reachable by k or less transitions from i, Tran' is the subset of transitions Tran which are reachable by k or less transitions, L' is the subset of labels $a \in L$ for which there is a transition $(s, a, s') \in \mathit{Tran}'$, and $I' = I \cap (\mathit{Tran}' \times \mathit{Tran}')$.

Let $f = (\sigma, \lambda) : T_0 \to T_1$ be a morphism of of $\mathbf{pTI}$. Define $f^{(k)}$, for $k \in \omega$, to be (σ', λ') where σ' is the restriction of σ to the states of $T_0^{(k)}$ and λ' is the restriction of λ to the labels of $T_0^{(k)}$.

This provides the appropriate definition of the family of functors $(-)^{(k)} : \mathbf{pTI} \to \mathbf{pTI}$ with which to generalise the argument of part (a). (This really is a generalisation once we regard an ordinary transition system as having an empty independence relation.)

Say an operation F on objects in $\mathbf{pTI}$ is definable if it acts on T so that

$$F(T) = \mathbf{pTI}[\![t]\!]\rho[T/x]$$

for some choice of process term t and variable x. Any such definable operation is $\trianglelefteq$-continuous and can be extended to an endofunctor on the subcategory of **pTI** in which morphisms are label-preserving. It will also satisfy

$$(F(T))^{(k)} = (F(T^{(k)}))^{(k)}, \text{for } k \in \omega,$$

—just as in part (a). As there,

$$(a(T))^{(k)} = (a(T^{(k-1)}))^{(k)}$$

so that any operation F defined by a term in which the variable is guarded satisfies

$$(F(T))^{(k)} = (F(T^{(k-1)}))^{(k)},$$

for $k > 0$. Given $f : T \to T'$ a morphism in **TI**, for $k \in \omega$, the morphism $f^{(k)} : T^{(k)} \to T'^{(k)}$ is the restriction of f to the states and labels of $T^{(k)}$. It follows that a functor F definable by a process term satisfies

$$(F(f))^{(k)} = (F(f^{(k)}))^{(k)},$$

on label-preserving morphisms f, for $k \in \omega$, and when associated with a guarded variable,

$$(F(f))^{(k)} = (F(f^{(k-1)}))^{(k)}.$$

We can now proceed as in part (a), reading "**pTI**" and "object of **pTI**" instead of "**T**" and "transition system"; the proofs of proposition 94 and lemma 9 are sufficiently abstract to apply equally well in the more general situation of **pTI**. This furnishes a proof of the uniqueness property of guarded recursions in **pTI**:

Lemma 95 *Suppose the variable x is guarded in t. Let T be an object of* **pTI** *and ρ be an environment in* **pTI**. *If $T \cong \mathcal{R}(\mathbf{pTI}[\![t]\!]\rho[T/x])$, a label-preserving isomorphism, then $T \cong \mathcal{R}(\mathbf{pTI}[\![rec\, x.t]\!]\rho)$, a label-preserving isomorphism.*

The next stage is to show:

Theorem 96 *Let t be a closed process term. Then, for an arbitrary environment ρ in* **pTI**,

$$\mathcal{O}p(t) \cong \mathcal{R}(\mathbf{pTI}[\![t]\!]\rho),$$

a label-preserving isomorphism.

Proof: The proof is by structural induction on terms t, with free variables $\overline{x}$, that for all closed terms $\overline{s}$ chosen as substitutions for the variables $\overline{x}$,

$$\mathcal{O}p(t[\overline{s/x}]) \cong \mathcal{R}(\mathbf{pTI}[\![t]\!]\rho[\overline{\mathcal{O}p(s)/x}]),$$

a label-preserving isomorphism. The operational rules have been chosen to make the cases of the induction straightforward for all but that of recursive processes—recourse is made to lemma 10 expressing acyclicity of the transition relation got operationally. The verification of the case of recursion proceeds as in the proof of theorem 11, relying on the uniqueness lemma 95 above. $\square$

We now finally obtain theorem 87 as a corollary, because, as remarked, the denotations $\mathbf{pTI}[\![t]\!]\rho$ and $\mathbf{TI}[\![t]\!]\rho$ coincide when t is closed.

References

[1] Abramsky, S., and Jagadeesan, R., Games and full completeness for multiplicative linear logic. Technical Report DoC 92/24, Imperial College, 1992.

[2] Advanced course on Petri nets, *Springer LNCS 254, 255*, 1987.

[3] Badouel, E. and Darondeau, P., Structural operational specifications and trace automata, in *Cleaveland, W.R. (ed.), Concur '92, Springer LNCS 630*, pp. 302–316, 1992.

[4] Barr, M. and Wells, C., *Category theory for Computer Science*. Prentice Hall, 1990.

[5] Bednarczyk, M.A., *Categories of asynchronous systems*, PhD thesis in Computer Science, University of Sussex, report no.1/88, 1988.

[6] Bénabou, J., Fibred categories and the foundations of naive category theory, *JSL 50*, pp. 10–37, 1985.

[7] Bergstra, J.A., and Klop, J.W., Process algebra for communication and mutual exclusion. Revised version, Report CS-R8409, Centrum voor Wiskunde en Informatica, Amsterdam, 1984.

[8] Berry, G., *Modèles completement adéquats et stables des λ-calculs typées*, Thèse de Doctorat d'Etat, Université Paris VII, 1979.

[9] Bloom, B., and Kwiatkowska, M., Trade-offs in true concurrency: Pomsets and Mazurkiewicz traces, *Brookes, S., Main, M., Melton, A., Mislove, M., and Schmidt, D., (eds), Mathematical Foundations of Programming Semantics*, Springer LNCS 598, pp. 350–375, 1992.

[10] Boudol, G., Atomic actions, note, *Bulletin of the European Association for Theoretical Computer Science, 38*, pp. 136–144, 1989.

[11] Boudol, G., Flow event structures and flow nets, in *I. Guessarian, (ed.), Semantics of Systems of Concurrent Processes, Springer LNCS 469*, pp. 62–95, 1990.

[12] Boudol, G. and Castellani, I., A non-interleaving semantics for CCS based on proved transitions, *Fundamenta Informaticae, XI, 4*, pp. 433–452, 1988.

[13] Boudol, G. and Castellani, I., Flow models of distributed computations: three equivalent semantics for CCS, *INRIA Research Report 1484*, 1991. To appear in Information and Computation.

[14] Brookes, S.D., On the relationship of CCS and CSP, in *Diaz, J. (ed.), Icalp '83, Springer LNCS 154*, pp. 83–96, 1983.

[15] Brookes, S.D., On the axiomatic treatment of concurrency, in *Brookes, Roscoe, Winskel (eds.), Seminar on Concurrency, Springer LNCS 197*, pp. 1–34, 1985.

[16] Brookes, S.D., Roscoe, A.W., and Walker, D.J., An Operational Semantics for CSP, submitted for publication.

[17] Brown, C. and Gurr, D., A linear specification language for Petri nets, accepted for *Mathematical Structures in Computer Science*.

[18] Camilleri, J., An operational semantics for occam, *International Journal of Parallel Programming, 18(5)*, 1989.

[19] Campbell, R.H. and Habermann, A.N., The specification of process synchroniation by path expressions, in *Gelenbe, E. and Kaiser, C., Operating Systems, Springer LNCS 16*, pp. 89–102, 1974.

[20] Castellani, I. and Zhang, G.Q., Parallel product of event structures, *Rapports de Recherche 1078, INRIA*, 1989.

[21] Curien, P-L., Categorical combinators, sequential algorithms, and functional programming, Birkhäuser, 1993.

[22] Degano, P., De Nicola, R. and Montanari, U., Observational equivalences for concurrency models, in *Wirsing, M. (ed.), Formal Description of Programming Concepts – III, IFIP*, Elsevier Science Publishers B.V., pp. 105–132, 1987.

[23] Degano, P., De Nicola, R. and Montanari, U., On the consistency of 'truly concurrent' operational and denotational semantics, extended abstract, in *IEEE Third Annual Symposium on Logic in Computer Science, Computer Society Press*, pp. 133–141, 1988.

[24] Droste, M., Event structures and domains, *Theoretical Computer Science, 68*, pp. 37–47, 1989.

[25] Van Glabbeek, R.J., *Comparative concurrency semantics and refinement of actions*, PhD thesis, CWI Amsterdam, 1990.

[26] Van Glabbeek, R.J., The linear time - branching time spectrum, in *Baeten, J.C.M. and Klop J.W. (eds.), Concur 90, Springer LNCS 458*, pp. 278–297, 1990.

[27] Van Glabbeek, R.J., The liinear time - branching time spectrum II, in *Best, E. (ed.), Concur 93, Springer LNCS 715*, pp. 66–81, 1993.

[28] Gorrieri, R., A hierarchy of system descriptions via atomic linear refinement, *Fundamenta Informaticae, 16*, pp. 289–336, 1992.

[29] Goubault, E. and Jensen, T.P., A homology of higher dimentional automata, in *Cleaveland, W.R. (ed.), Concur 92, Springer LNCS 630*, pp. 254–268, 1992.

[30] Grabowski, J., On partial languages, *Fundamenta Informaticae, IV, 2*, pp. 427–498, 1981.

[31] Grothendieck, A., Catégories fibrées et descente, in *Grothendieck, A., (ed), Revêtement étales et groupe fondamental, (SGA 1), Exposé VI*, Springer Lecture Notes in Mathematics 224, 1971.

[32] Gunawardena, J., Causal Automata, *Theoretical Computer Science, 101*, pp. 265–288, 1992.

[33] Hoare, C.A.R., A model for communicating sequential processes, *Technical Report PRG-22, Programming Research Group, University of Oxford Computing Lab.*, 1981.

[34] Hoare, C.A.R., Brookes, S.D. and Roscoe, A.W., A Theory of Communicating Processes, *JACM 31,3*, pp. 560–599, 1984.

[35] Hennessy, M., *Algebraic theory of processes*, MIT Press, 1988.

[36] Hyland,J.M.E., and Ong, C-H.L., Fair Games and Full Completeness for Multiplicative Linear Logic without the MIX-rule. Working draft to appear as a report of the Computer Laboratory, University of Cambridge, 1993.

[37] Jacobs, B.P.F., *Categorical type theory*, PhD thesis in Mathematics, University of Nijmegen, 1991.

[38] Johnstone, P., Fibred categories. *Lecture notes, Cambridge University*, 1983.

[39] Joyal, A., Nielsen, M. and Winskel, G., Bisimulation and open maps, *Proc. of LICS 93*, pp. 418–427, 1993.

[40] Kahn, G. and Plotkin, G., Structures de données concrètes, *IRIA-Laboria Report 336*, 1979.

[41] Katz, S. and Peled, D., An efficient verification method for parallel and distributed programs, in *de Bakker, de Roever and Rozenberg (eds.), Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency, Springer LNCS 354*, pp. 489–507, 1989.

[42] Keller, R.M., Formal verification of parallel programs, *Communications of the ACM, no.19, vol.7*, pp. 371–384, 1976.

[43] Kwiatkowska, M., *Categories of Asynchronous Systems*, PhD thesis, University of Leicester, 1989.

[44] Labella, A. and Petterossi, A., Categorical models of process cooperation, in *Pitt, Abramsky, Poigné and Rydeheard (eds.), Category Theory and Computer Programming, Springer LNCS 240*, pp. 282–298, 1985.

[45] Lamarche, F., Sequentiality, games and linear logic, *Proceedings of CLICS Workshop–Part I & II, Aarhus, March 1992* DAIMI PB 398, Aarhus University, 1992.

[46] Lamport, L., Time, clocks and the ordering of events in a distributed system, *CACM, 21*, pp. 558–565, 1978.

[47] Larsen, K. and Skou, A., Bisimulation through probabilistic testing, *in Principles of Programming Languages*, pp. 344–351, 1989.

[48] Lauer, P. and Campbell, R.H., Formal semantics for a class of high level primitives for coordinating concurrent processes, *Acta Informatica, 5*, pp. 297–332, 1975.

[49] Langerak, R., Bundle event structures: A non-interleaving semantics of LOTOS, *Memoranda Informatica 91–60*, University of Twente, 1991.

[50] MacLane, S., *Categories for the Working Mathematician, Graduate Texts in Mathematics*, Springer, 1971.

[51] Manna, Z. and Pnueli, A., *The temporal Logic of Reactive and Concurrent Systems*, Springer Verlag, 1991.

[52] Martin-Löf, P., The domain interpretation of type theory, Lecture notes, Göteborg, 1983.

[53] Meseguer, J., and Montanari, U., Petri nets are monoids: a new algebraic foundation for net theory, *Proc. of LICS 88*, pp. 155–164, 1988.

[54] Milner, A.R.G., Calculus of communicating systems, *Springer LNCS 92*, 1980.

[55] Milner, A.R.G., *Communication and concurrency*, Prentice Hall, 1989.

[56] Milner, A.R.G., Calculi for synchrony and asynchrony, *Theoretical Computer Science, 25*, pp. 267–310, 1983.

[57] Mukund, M., A transition system characterization of Petri nets, *Report TCS-91-2*, School of Mathematics, SPIC Science Foundation, 1990.

[58] Mukund, M. and Thiagarajan, P.S., An axiomatization of event structures, in *Veni Madhavan, C.E. (ed), FST & TCS 89, Springer LNCS 405*, pp. 143–160, 1989.

[59] Mukund, M. and Thiagarajan, P.S., An axiomatization of well branching prime event structures, *Report TCS-90-2*, School of Mathematics, SPIC Science Foundation, 1990.

[60] Mukund, M. and Nielsen, M., CCS, locations and asynchronous transition systems, in *Shyamasundar, R. (ed.), FST & TCS 92, Springer LNCS 652*, pp. 328–341, 1992.

[61] Mazurkiewicz, A., Concurrent program schemes and their interpretations, *DAIMI PB-78, Computer Science Department, University of Aarhus*, 1977.

[62] Mazurkiewicz, A., Basic notions of trace theory, in *in de Bakker, de Roever and Rozenberg (eds.), Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency, Springer LNCS 354*, pp. 285–363, 1988.

[63] Nicollin, X. and Sifakis, J., An overview and synthesis on timed process algebras, *Springer LNCS 575*, pp. 376–398, 1992.

[64] Nielsen, M., Plotkin, G. and Winskel, G., Petri nets, Event structures and Domains, part 1, *Theoretical Computer Science, vol. 13*, pp. 85–108, 1981.

[65] Nielsen, M., Rozenberg, G. and Thiagarajan, P.S., Elementary transistion systems, *Theoretical Computer Science 96*, pp. 3–33, 1992.

[66] Ochmanski, E., Regular behaviour of concurrent systems, *Bulletin of the European Association for Theoretical Computer Science (EATCS), 27*, pp. 56–67, 1985.

[67] Olderog, E. and Hoare, C.A.R., Specification oriented semantics for communicating processes, *Diaz, J. (ed.), Icalp '83, Springer LNCS 154*, pp. 561–572, 1983.

[68] Olderog, E.-R., *Nets, Terms and Formulas. Three views of Concurrent Processes and Their Relationships*, Cambridge University Press, 1991.

[69] Peirce, B.C., Category theory for computer scientists, in *the Foundations of Computing Series*, The MIT Press, 1991.

[70] Penczek, W., A temporal logic for event structures, *Fundamenta Informaticae, XI* pp. 297–326, 1988.

[71] Petri, C.A., Non-sequential processes, GMD-ISF Report ISF-77-05, 1977.

[72] Pinna, G.M. and Poigné, A., On the nature of Events, in *Havel, I.M. and Koubek, V. (eds.), Mathematical Foundations of Computer Science 1992*, Springer LNCS 629, pp. 430–441, 1992.

[73] Pinter, S. and Wolper, P., A temporal logic for reasoning about partially ordered computations, *Proc. 3rd ACM PODC*, pp. 24–37, 1984.

[74] Plotkin, G.D., Structural operational semantics, *Lecture Notes, DAIMI FN-19, Computer Science Department, University of Aarhus*, 1981 (reprinted 1991).

[75] Poigné, A., Category theory and logic, in *Pitt, Abramsky, Poigné and Rydeheard (eds.), Category Theory and Computer Programming, Springer LNCS 240*, pp. 252–265, 1985.

[76] Pratt, V.R., Modelling concurrency with partial orders, *International Journal of Parallel Programming, 15,1*, pp. 33–71, 1986.

[77] Pratt, V.R., Modelling concurrency with geometry, in *Proc. 18th Ann. ACM on Principles of Programming Languages*, pp. 311–322, 1991.

[78] Rabinovich, A. and Trakhtenbrot, B.A., Behaviour Structure and Nets, *Fundamenta Informaticae, XI, 4* pp. 357–404, 1988.

[79] Rensink, A., *Models and Methods for Action Refinement*, PhD thesis University of Twente, 1993.

[80] Rozoy, B. and Thiagarajan, P.S., Event structures and trace monoids, *Theoretical Computer Science, 91, 2*, pp. 285–313, 1991.

[81] Sassone, V., Nielsen, M. and Winskel, G., A classification of models for concurrency, in *Best, E.(ed.), Concur '93, Springer LNCS 715*, pp. 82–96, 1993.

[82] Sassone, V., Nielsen, M. and Winskel, G., Deterministic behavioural models for concurrency, in *Borzyszkowski, A.M., and Sokolowski, S., (eds), MFCS 93, Springer LNCS 711*, pp. 682–692, 1993.

[83] Scott, D.A., Domains for denotational semantics, in Nielsen, M. and Schmidt, E.M. (eds.): *Automata, Languages and Programming, Springer LNCS 140*, pp. 577–613, 1982.

[84] Shields, M.W., Concurrent machines, *Computer Journal, vol. 28*, pp. 449–465, 1985.

[85] Stark, E.W., Concurrent Transition Systems, *Theoretical Computer Science, 64*, pp. 221–269, 1989.

[86] Stirling, C., Modal and temporal logics, *in Handbook of Logic in Computer Science, vol. I, edited by Abramsky, S., Gabbay D. and Maibaum, T., Oxford University Press*, 1992.

[87] Taylor, P., *Recursive domains, indexed category theory and polymorphism*, PhD thesis, Cambridge University, 1987.

[88] Thiagarajan, P.S., Elementary net systems, in *Brauer, Reissig and Rozenberg (eds.), Petri Nets: Central models and their properties, Springer LNCS 254*, pp. 26–59, 1987.

[89] Thomas, W., On logical definability of trace languages, in V. Diekert, ed., *Proceedings of a workshop of the ESPRIT Basic Research Action no 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS), Kochel am See, Bavaria, FRG*, Report TUM 19002, Technical University of Munich, pp. 172–182, 1990.

[90] Vogler, W., Bisimulation and action refinement, *Theoretical Computer Science, 114*, pp. 173–200, 1993.

[91] Winskel, G., Events in Computation. *PhD thesis, University of Edinburgh, available as a Comp. Sc. report*, 1980.

[92] Winskel, G., Event structure semantics of CCS and related languages, in *Nielsen, M. and Schmidt, E.M. (eds), Icalp '82, Springer LNCS 140*, pp. 561–576, 1982. A full version with proofs appears as DAIMI PB-159, Computer Science Department, University of Aarhus, 1983.

[93] Winskel, G., A representation of completely distributive algebraic lattices, *Report of the Computer Science Department, Carnegie-Mellon University*, 1983.

[94] Winskel, G., Synchronisation trees, *Theoretical Computer Science, 34*, pp. 33–82, 1985.

[95] Winskel, G., Categories of Models for Concurrency, in *Brookes, Roscoe and Winskel (eds.), Seminar on Concurrency, Springer LNCS 197*, pp. 246–267, 1984.

[96] Winskel, G., Petri nets, algebras, morphisms and compositionality, *Information and Computation, 72*, pp. 197–238, 1987.

[97] Winskel, G., Event structures, in *Brauer, Reissig and Rozenberg (eds.), Petri Nets: Applications and relationships to othe models of concurrency, Springer LNCS 255*, pp. 325–392, 1987.

[98] Winskel, G., An introduction to event structures, in *de Bakker, de Roever and Rozenberg (eds.), Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency, Springer LNCS 354*, pp. 364–397, 1988.

[99] Winskel, G., A category of labelled Petri nets and compositional proof system, *Proc. of LICS 88*, pp. 142–154, 1988.

[100] Winskel, G., A compositional proof system on a category of labelled transition systems, *Information and Computation, vol. 87*, pp. 2–57, 1990.

[101] Winskel, G., *The Formal Semantics of Programming Languages. An Introduction*, The MIT Press, 1993.

[102] Wolper, P. and Godefroid, P., Partial-order methods for temporal verification, in *Best, E. (ed.), Concur '93, Springer LNCS 715*, pp. 233–246, 1993.

[103] Zielonka, W., Safe executions of recognizable trace languages by asynchronous automata, in Mayer, A.R. et al. (eds.), *Proc. Symposium on Logical Foundations of Computer Science, Logic at Botik '89, Pereslavl-Zalessky (USSR), Springer LNCS 363*, pp. 278–289, 1989.