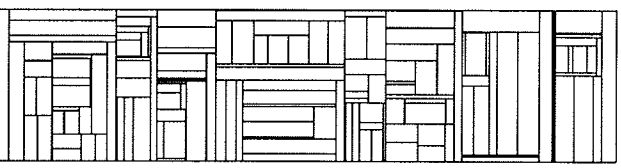
ISSN 0105-8517

Models for Concurrency

Glynn Winskel
Mogens Nielsen

DAIMI PB – 429
November 1992

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark



MODELS FOR CONCURRENCY

(Draft copy of November 27, 1992)

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

This is a draft version of a chapter for the Handbook of Logic and the Foundations of Computer Science, Oxford University Press. It surveys a range of models for parallel computation to include interleaving models like transition systems, synchronisation trees and languages (often called Hoare traces in this context), and models like Petri nets, asynchronous transition systems, event structures, pomsets and Mazurkiewicz traces where concurrency is represented more explicitly by a form of causal independence. The presentation is unified by casting the models in a category-theoretic framework. One aim is to use category theory to provide abstract characterisations of constructions like parallel composition valid throughout a range of different models and to provide formal means for translating between different models. It is very much a draft at present. In particular, the "Notes" surveying related work are incomplete and the appendix on fibred categories needs to be overhauled in the light of some slick proofs, provided by Bart Jacobs. It is ragged in other places too. Constructive comments and corrections will be appreciated.

A knowledge of basic category theory is assumed, up to an acquaintance with the notion of adjunction. A good reference is [18].

Contents

Chapter 1

Introduction

The purpose of this work is to provide a survey of some of the fundamental models for distributed computations, used and studied within theoretical computer science.

The general nature of such a model is a mathematical formalism in which to represent and reason about the behaviour of distributed computational systems. The purpose of such models is to provide a conceptual understanding of systems and their behaviour in theory, and to contribute to the design and analysis methods applied in practice.

In the rich theory of sequential computational systems, several mathematical formalisms have been introduced and studied in depth, *e.g.* Turing Machines, Lambda Calculus, Post Systems, Markov Systems, Random Access Machines, etc. One main result from this theory, is that these formalisms are all equivalent, in the sense that their associated classes of behaviours in terms of input-output functions are all the same.

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

Hence, in the study of reactive systems, one is forced to take a different and less abstract view of behaviours than the traditional functional one. One needs a notion of behaviour, expressing aspects of the patterns of actions, which a system is capable of performing. Such aspects could include well-known phenomena like deadlock, mutual exclusion, starvation, etc.

One may see the role of such models in computer science as providing the foundation for the development of all other theoretical and practical research areas on distributed computing. To give some examples, they are used to provide the semantics of process description languages, and hence the basis of the many behavioural equivalences studied in the literature on prscess calculi. They are used to give the formal definition of specification logics, and hence the basis for 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 such models have been suggested and studied over the last 10-15 years. Here we shall survey some of these, but we shall make no attempt to come even close

to a complete survey. Rather, we have chosen to present in some detail a few carefully selected models.

Common to all of these models, is that they rest on the important idea of atomic actions, over which the behaviour of a system is defined. The models differ mainly with respect to how detailed such behaviors of systems may be 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 here is to present prime representatives of these classes of models, covering the landscape of models from the most abstract to the most concrete, and to formalise and study the exact nature of the relationships between these models, representing fundamental abstraction steps. In other words we would like to set the scene for a formal classificaion 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).

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

Imagine further 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 made in the classification of models is the one between the so-called interleaving and noninterleaving models. The main characteristics of an interleaving model is that it will abstract away from the fact that our system is actually composed of three independently computing agents, and model the behaviour in terms of purely sequential patterns of actions. Formally, the behaviour of our system will be expressed in terms of some kind of nondeterministic merging or interleaving of the sequential behaviours of the three components. Prime examples of such models are Transition Systems [15]], Synchronization Trees [21], Acceptance Trees [13], and Hoare Traces [10].

It is important to realise that in many situations this kind of abstraction is exactly what one wants, and it has fully 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 of no relevance for the things you would like to reason about.

However, there may be situations in which it could be important to keep information about the fact that our system is composed of the three independently computing components, a possibility offered by the so-called noninterleaving models, with Petri Nets [1], Event Structures [37], and Mazurkiewicz Traces [20] as prime examples. One such situation is that some behavioural properties may rest on the fact that each component is a separate physical entity with its own progress of computation, typically so-called liveness properties. Dealing with such properties in interleaving models is often handled by some specific coding of the actions belonging to the components combined with some logical assertions expressing progress assumptions for the system in question, *i.e.* handled outside the actual 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 Medium is actually erroneous, in the sense that delivering a message from the Sender may leave the Medium in either a normal state, having acepted 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 restricted behaviour of the Medium (and hence from some possibilities of deadlocks). Formally, these models typically express the total nondeterministic behavior of our system in terms of the set of possible (determinate) "runs" of the system. Prime examples are Hoare Traces, Mazurkiewicz Traces and Pomsets [27].

As indicated, in many situations a more detailed representation of when nondeterministic choices are made during a computation is necessary, to express possible deadlocks and other safety properties of systems. And this is possible to various degrees in branching time models like Synchronization and Acceptance Trees, Petri Nets, and Event Structures. Of course, the treatment of nondeterminism is particularly important for the interleaving models, where also the notion of independent activities is expressed in terms of nondeterminism.

Finally, yet a third distinction is made between models allowing an explicit representation of the (finite) states in the implementation of the components of our system, and models abstracting away from such information focussing purely on behaviour in terms of patterns of occurrences of actions during time. Prime examples of the first type are Transition Systems and Petri Nets, and of the second type Trees, Event Structures and Traces.

So, to sum up the seemingly confusing world of models for concurrency has some structure in the form of a classification according to expressiveness, and one of our main goals will be to formalise this structure. In the process we shall follow another central theme of our chapter: the use of category theory as a convenient language for formalising relationships and transfer of techniques among models.

The main idea is that each model will be equipped with a notion of (behaviour preserving) morphism, making it into a category. As we shall see, it turns out that certain types of adjunctions (reflections and coreflections) are just the right abstract way of expressing the fact that one model is embedded in (more abstract than) another, even when the two models are expressed in very different mathematical terminologi. One part of the adjunction will tell how to embed the more abstract model into the other, the other part will formally abstract away from information, as indicated intuitively above.

Such categorical adjunctions not only provide an aid in the understanding of the different models and their relationships, but also an important vehicle for the transfer of techniques from one model to another. In this chapter we shall present a number of such results, focussing on the role of models in giving formal semantics of process description languages. We shall see how basic operations of such languages may be understood as universal constructs (like product and coproduct) of our categories. Since such universal constructs are defined up to isomorphism, this provides us with some general guidelines and justification of the definition of operations. They come as universal constructs when viewing models categorically, and not as arbitrary *ad hoc* definitions. And, importantly, from this and general facts of category theory they may be transferred between models via the established adjunctions.

Our goal is to survey a few but fundamental models for concurrency, and exploit the

use of category theory as a language for these models and their relationship.

A knowledge of basic category theory is assumed, up to an acquaintance with the notion of adjunction. A good reference is [18]. The work uses a particular representation of partial functions, the details and notation for which are found in Appendix B. One warning as regards terminology: we use the term "coreflection" to mean an adjunction in which the unit is a natural isomorphism. Similarly, "reflection" is used here to mean an adjunction for which the counit is a natural isomorphism.

Chapter 2

Interleaving models

2.1 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.

2.1.1 A category of transition systems

Transition systems are a frequently used model of parallel processes. They 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

$$(S, i, L, Tran)$$

where

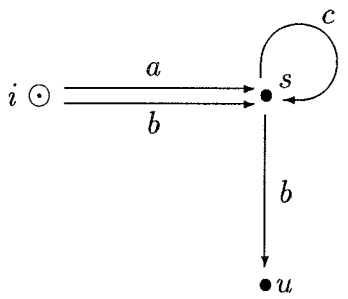
- S is a set of *states* with *initial state* i,
- L is a set of *labels*, always assumed to not contain a distinguished symbol $*$,
- $Tran \subseteq S \times L \times S$ is the *transition relation.*

This definition narrows attention to transition systems, which are *extensional* in that they cannot have two distinct transitions with the same label and 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 $(s, a, s') \in Tran$.This notation lends itself to the familiar graphical notation for transition systems. For example,



represents a transition system which at the initial state i 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 shall write

$$s \not\xrightarrow{a}$$

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

It is technically convenient to introduce *idle* transitions, associated with any state.

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

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

Idle transitions play a role in the definition of morphism between transition systems.

Definition: Let

$$T_0 = (S_0, i_0, L_0, Tran_0) \text{ and}$$
$$T_1 = (S_1, i_1, L_1, Tran_1)$$

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

$$(s, a, s') \in Tran_0 \Rightarrow (\sigma(s), \lambda(a), \sigma(s')) \in Tran_{1*}.$$

With the introduction of a idle transitions, morphisms on transition systems can be described as preserving transitions and the initial state. Observe that morphisms between labelled transition system can be characterised in a way which does not involve idle transitions. According to this characterisation a morphism between two transition systems $T_0 = (S_0, i_0, L_0, Tran_0)$ and $T_1 = (S_1, i_1, L_1, Tran_1)$ consists of (σ, λ), where
$\sigma : S_0 \to S_1$ and $\lambda : L_0 \to_* L_1$, which satisfy

$$\sigma(i_0) = i_1'$$
$$(s, a, s') \in Tran_0 \ \&\ \lambda(a) \text{ defined } \Rightarrow (\sigma(s), \lambda(a), \sigma(s')) \in Tran_1, \text{ and}$$
$$(s, a, s') \in Tran_0 \ \&\ \lambda(a) \text{ undefined } \Rightarrow \sigma(s) = \sigma(s').$$

As this characterisation makes clear, 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, a device which avoids the fuss of considering whether or not $\lambda(a)$ is defined. It is to be 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 with morphisms as defined form a category which will be the first important category in our study:

Proposition 2 *Transition systems with morphisms form a category in which the composition of two morphisms $f = (\sigma, \lambda) : T_0 \to T_1$ and $g = (\sigma', \lambda') : T_1 \to T_2$ is $g \circ f = (\sigma' \circ \sigma, \lambda' \circ \lambda) : T_0 \to T_2$ 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 and 1_L is the identity function on the labelling set L of T.*
(Here composition on the left of a pair is that of total functions while that on the right is of partial functions.)

Definition: Denote by **T** the category of labelled transition systems given by the last proposition.

2.2 Constructions on transition systems

The category of transition systems is rich in categorical constructions which furnish the basic combinators for languages of parallel processes.

2.2.1 Restriction and relabelling

Restriction and relabelling are important operations 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. 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.

Restriction and relabelling are constructions which depend on labelling sets and functions between them. Seeing them as categorical constructions involves dealing explicitly with functions on labelling sets and borrowing a couple of fundamental ideas from fibred category theory. (In fact, restriction and relabelling arise as *cartesian* and *cocartesian* morphisms respectively.) Consider restriction first.

Definition: Let $T' = (S, i, L', Tran')$ be a transition system. Let $\lambda : L \hookrightarrow L'$ be an inclusion morphism. Define the *restriction* $\lambda^*(T')$ to be the transition system $(S, i, L, Tran)$ with

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

The operation of restriction simply cuts out those transitions with labels not in the restricting set. To understand it as a universal construction we observe that there is a functor $p : \mathbf{T} \to \mathbf{Set}_*$, to 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 $\lambda^*(T')$ is a morphism $f : \lambda^*(T') \to T'$, 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 $h : T \to \lambda^*(T')$ such that $p(h) = 1_L$ and $f \circ h = g$.

This says that the "inclusion" morphisms associated with restrictions are *cartesian liftings* of inclusion maps between labelling sets. In fact, they are strong cartesian—see Appendix A.

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

The operation of relabelling is associated with a dual construction, that of forming a *cocartesian lifting*. When $\lambda : L \rightarrow_* L'$ is total, the relabelling construction $\lambda_!$ takes a transition system T with labelling set L to $\lambda_!(T)$, the same underlying transition system but relabelled according to λ.

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

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

Letting the transition system T have states S, there is a morphism $(1_S, \lambda) : T \rightarrow \lambda_!(T)$. 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 $h : \lambda_!(T) \rightarrow T'$ such that $p(h) = 1_{L'}$ and $h \circ f = g$.

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 $f : T \rightarrow \lambda_!(T)$, given by $f = (1_S, \lambda)$ which is strong cocartesian—see Appendix A.*

Remark: In fact there are strong cartesian liftings for any $\lambda : L \rightarrow_* L'$ and any X' with labelling set L', and the functor $p : \mathbf{T} \rightarrow \mathbf{Set}_*$ is a fibration. It is also a cofibration, and thus a bifibration, taking advantage of the fact that the relabelling construction can also be defined when λ is partial.

2.2.2 The nil transition system

The *nil* transition system

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

the transition system consisting of a single initial state i, is initial and terminal in the category of transition systems.

2.2.3 The product of transition systems

The product in the category of labelled transition systems is central to representing on transition systems the parallel compositions of processes of languages, which like CCS, CSP and Occam, communicate by events of synchronisation.

Definition: Assume transition systems $T_0 = (S_0, i_0, L_0, Tran_0)$ and $T_1 = (S_1, i_1, L_1, Tran_1)$. 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 $\rho_0 : S_0 \times S_1 \rightarrow S_0$, $\rho_1 : S_0 \times S_1 \rightarrow S_1$
- $L = L_0 \times_* L_1 =$
  $\{(a, *) \mid a \in L_0\} \cup \{(*, b) \mid b \in L_1\} \cup \{(a, b) \mid a \in L_0, b \in L_1\}$,
  with projections π_0, π_1, and
- $(s, a, s') \in Tran_* \Leftrightarrow$
  $(\rho_0(s), \pi_0(a), \rho_0(s')) \in Tran_{0*}$ & $(\rho_1(s), \pi_1(a), \rho_1(s')) \in Tran_{1*}$.

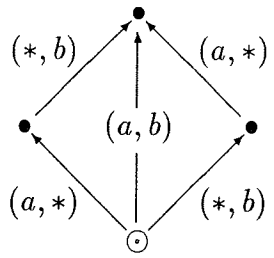
Define $\Pi_0 = (\rho_0, \pi_0)$ and $\Pi_1 = (\rho_1, \pi_1)$.

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:



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 transitions 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 perhaps disallow their absences) and relabelling (to rename the results of synchronisations). In special cases, such as parallel compositions of versions of CSP, the construction can be streamlined (see the section on "Parallel compositions").

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

We remark that the product-machine contruction from automata theory arises as a *fibre product, viz.* a product in a fibre. A fibre $p^{-1}(L)$ consists of transition systems with a common labelling set L, in which the morphisms are precisely those f for which $p(f) = 1_L$.

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 as a result of first taking a product of T_0 and T_1, with labelling sets L_0, L_1 respectively, to give a transition system

$T_0 \times T_1$ with labelling set $L_0 \times_* L_1$, then restricting by taking $i^*(T_0 \times T_1)$ for an inclusion $i : S \to L_0 \times_* L_1$, followed by a relabelling $r_!(i^*(T_0 \times T_1))$ with respect to a total $r : S \to_* L$. 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. 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 $r_! i^*(T_0 \times T_1)$ where $i : D \to L \times_* L$ 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 [32, 34].

Example: *The synchronisation algebra for pure CCS:* In CCS [22] 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
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdots$	$\cdot$	$\cdot$

Example: *The synchronisation algebra for $\|$ in TCSP:* In "theoretical" CSP—see [11, 6]—events are labelled by $a, b, \cdots$ or τ. For one of its parallel composition ($\|$) 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 Sums

Nondeterministic sums arise from coproducts. The category of transition systems has coproducts:

Definition: *The coproduct of transition systems*
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
- $L = L_0 \uplus L_1$ with injections j_0, j_1
- $$t \in Tran \Leftrightarrow \exists (s, a, s') \in Tran_0.\ t = (in_0(s), j_0(a), in_0(s')) \text{ or}$$
$$\exists (s, a, s') \in Tran_1.\ t = (in_1(s), j_1(a), in_1(s')).$$

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 for example CCS. We look to coproducts in the fibres.

Each fibre $p^{-1}(L)$ has coproducts for a labelling set L. 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 they are very similar to coproducts of transition systems in general—they differ only in the labelling part.

Definition: *The fibre coproduct of transition systems*
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. Their fibre coproduct $T_0 +_L T_1 = (S, i, L, Tran)$ (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

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

Fibre coproducts give coproducts in the fibres.

Proposition 7 *Let T_0 and T_1 be transition systems over L. The transition system $T_0 +_L T_1$ with injections $(\iota_0, 1_L)$ and $(\iota_1, 1_L)$, as defined above, is a coproduct in the subcategory of transition systems over 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 $T_0 \oplus T_1$ to be $(S, i, L_0 \cup L_1, \mathit{Tran})$ 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

$$t \in \mathit{Tran} \Leftrightarrow \exists (s, a, s') \in \mathit{Tran}_0.\ t = (in_0(s), a, in_0(s')) \text{ or}$$
$$\exists (s, a, s') \in \mathit{Tran}_1.\ t = (in_1(s), a, in_1(s')).$$

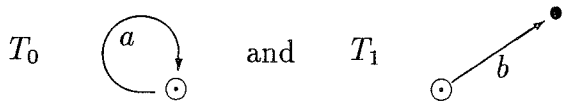
This sum can be understood as a fibre coproduct, but where first we form cocartesian liftings of the inclusion maps into the union of the labelling sets. This simply has the effect of enlarging the labelling sets to a common labelling set, their union, where we can form the fibre coproduct.

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*

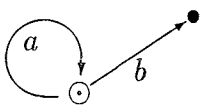
$$T_0 \oplus T_1 \cong j_{0!}T_0 +_{(L_0 \cup L_1)} j_{1!}T_1.$$

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. For a transition system $T = (S, i, L, Tran)$, the operation of prefixing involves a choice of element i', for the new starting state, and an injective function $\lfloor - \rfloor$ on S making copies of the old states so that they are distinct from i', *i.e.* so that $i' \neq \lfloor s \rfloor$ for any $s \in S$. This is achieved by, for instance, taking i' to be $\emptyset$ and $\lfloor s \rfloor = \{s\}$. Prefixing on transition systems, is defined concretely by:

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

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

Let $(\sigma, \lambda) : T_0 \to T_1$ be a morphism of transition systems where λ is a total function. Define $a(\sigma, \lambda)$ to be (σ', λ') where

$$\sigma'(s') = \begin{cases} \emptyset & \text{if } s' = \emptyset, \\ \{\sigma(s)\} & \text{if } s' = \{s\} \end{cases} \quad \text{and} \quad \lambda'(b) = \begin{cases} a & \text{if } b = a, \\ \lambda(b) & \text{otherwise.} \end{cases}$$

Prefixing $a(-)$ is a functor on the subcategory in which all functions on labelling sets are total, and in particular between fibres $p^{-1}(L) \to p^{-1}(L \cup \{a\})$. 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. The fact that functions between labelling sets can be partial causes trouble; how should the prefixing operation $a(-)$ act on the unique morphism $anil \to nil$?

2.3 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. A restriction $t \restriction \Lambda$ is understood to

denote $i^*(T)$, given by the cartesian lifting of $\Lambda \cap L \hookrightarrow L$ with respect to the denotation T of t, assumed to have labelling set L. This sensible also in the situation where the labelling set L of t 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. 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

$[\![nil]\!]\rho = nil$, for a choice of initial transition systems
$[\![t_0 \oplus t_1]\!]\rho = [\![t_0]\!]\rho \oplus [\![t_1]\!]\rho$, 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 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 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 $fix(F) =_{def} \bigcup_{n\in\omega} F^n(\bot)$. 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 have:

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 $T \cong \mathcal{R}(F(T))$, then $T \cong \mathcal{R}(fix(F))$.*

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

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

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

Operational semantics (version 1)

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

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

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

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

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

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 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.

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:

Operational semantics (version 2)

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

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

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

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

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

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

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 [22], 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. ∎

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

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

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

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

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 vars $\overline{x}$ 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

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

Hence

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

by the induction hypothesis. Thus $\mathcal{O}p(rec\ y.v) \cong \mathcal{R}(F(\mathcal{O}p(rec\ y.v))$ 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,

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

as required in this case for the induction.∎

2.4 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

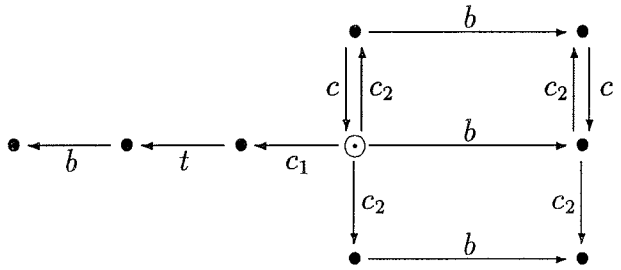
We introduce an example which will reappear in illustrating all the different models. In a form of process algebra it might be described by

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

where

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

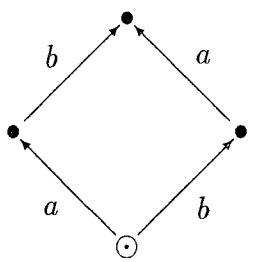
Intuitively, SYS consists of two vending machines VM, VM' in parallel with a customer C. The customer can insert a coin ($c_2!$) to get coffee ($c?$) repeatedly or insert a coin ($c_1!$) to get tea ($t?$) and stop. The vending machine VM can receive a coin ($c_2?$) to deliver coffee ($c!$) or alternatively receive ($c_2?$) to deliver tea ($t!$). 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 2.3, illustrated below:



Notice, in particular, the deadlock which can occur if the customer inserts coin c_2 in machine VM. 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:

$$a\; b\; nil + b\; a\; nil \qquad\qquad a\; nil \;||\; b\; nil.$$

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



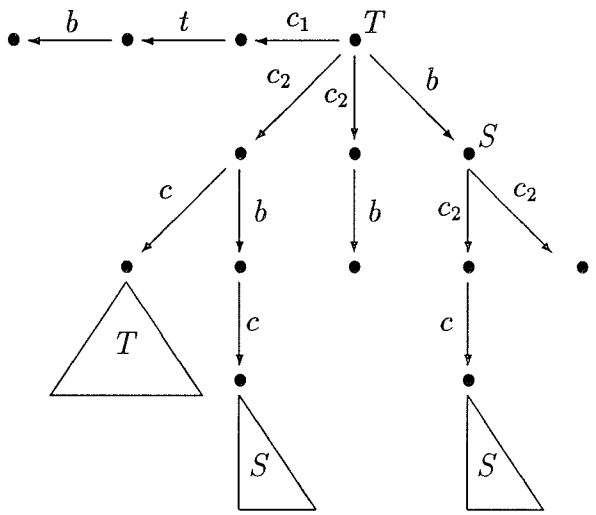
This contrasts the interleaving model of transition systems with noninterleaving models like Petri nets we shall see later, which represent independence of actions explicitly.

2.5 Synchronisation trees

We turn to consider another model. It is 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 [21], 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 2.4. As a synchronisation tree SYS could be represented by



where the trees stemming from S and T are repeated as indicated. The synchronisation trees is obtained by unfolding the transition-system of 2.4. 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 **S** for the full subcategory of synchronisation trees in **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.[1]

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

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

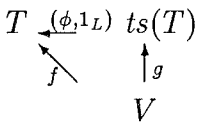
$$(t_1, \cdots, t_j, t_{j+1}, \cdots, t_{n-1})$$

  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 12 *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 $f = (\phi, 1_L) \circ g$:*



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 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 $\sigma_1(v') = \sigma(v)$ if $\lambda(a) = *$ and otherwise, in the case where $\lambda(a)$ is defined, take $\sigma_1(v') = \sigma(v)((\sigma(v), \lambda(a), \sigma(v'))$.

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 [34].) ∎

[1] 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.

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 [18] 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 [18] p.88).

Like transition systems, synchronisation trees have been used to give semantics to languages like CCS and CSP (see *e.g.* [21], [7]). 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 [21]). In our context, the expansion of a parallel composition as a nondeterministic sum appears as a characterisation of the product of synchronisation trees.

Proposition 13 *The product of two synchronisation trees S and T of the form*

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

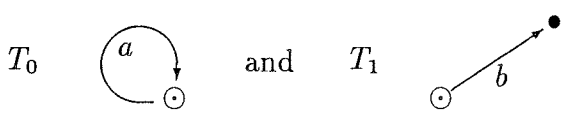
is given by

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

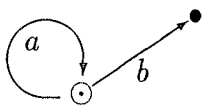
Proof: The fact that the category of synchronisation trees has products and that they are preserved by the unfolding operation ts is a consequence of the general fact that right adjoints preserve limits. In particular, $ts(S \times_T T)$ is a product of the synchronisation trees S and T above; the proof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

The adjunction between transition systems and synchronisation trees is fibrewise in that it restricts to adjunctions between fibres over a common labelling set. For this reason its right adjoint of unfolding automatically preserves restriction, while its left adjoint, the functor regarding a tree as a transition system, preserves relabelling (see Appendix A). 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



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 $ts(T_0) +_L ts(T_1)$.

This section indicates how the coreflection between synchronisation trees and labelled transition systems helps in formulating and proving the relationship between the different models. For labelled transition systems and synchronisation trees general facts like the existence of an adjunction or the preservation of cartesian morphisms by functors can be brought to bear on proofs showing, for instance, how semantics is preserved in passing from one model to another.

2.6 Languages

Synchronisation trees abstract away from the looping structure of processes. Now we examine a yet more abstract model, that of languages where we further abstract away from 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 () is always contained in H. Such languages were called *traces* in [10] and for this reason, in the context of modelling concurrency, they are sometimes called *Hoare traces*. They consist however of simply 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 2.4. 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

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

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

Ordering sequences 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 **L** to **S**. On the other hand any transition system, and in particular any synchronisation tree, gives rise to a language consisting of sequences of labels obtained from the sequences of transitions it can perform. This operation extends to a functor. The two functors form an adjunction from **S** to **L** (but not from **T** to **L**).

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

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

Let $T = (S, i, L, Tran)$ be a synchronisation tree. Define $sl(T) = (H, L)$ where a sequence $h \in L^*$ 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 14 *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 $\lambda = 1_L \circ sl(g)$:*

$$\begin{array}{ccc} (H, L) & \xleftarrow{1_L} & sl \circ ls(H, L) \\ & \nwarrow^{\lambda} & \uparrow{\scriptstyle sl(g)} \\ & & sl(T) \end{array}$$

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 $\lambda = 1_L \circ sl(g)$. ∎

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.

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.

Let $r : \mathbf{L} \to \mathbf{Set}_*$ be the functor sending a morphism $\lambda : (H, L) \to (H', L')$ of languages to $\lambda : L \to_* L'$. The expected constructions of restriction and relabelling arise as (strong) cartesian and cocartesian liftings.

2.7 Relating semantics

We can summarise the relationship between the different models by recalling the coreflection and reflection:

$$\mathbf{T} \overset{coref}{\leftarrow} \mathbf{S} \overset{ref}{\leftarrow} \mathbf{L}$$

The functors shown are essentially full inclusions. They each have adjoints; the inclusion of the coreflection has a right and that of the reflection a left adjoint. These functors from left to right correspond respectively to losing information about the looping,and then in addition the nondeterministic branching structure of processes.

Such categorical facts are useful in several ways. The coreflection between **S** and **T** tells us how to construct limits in **S** from those it **T**. In particular, we have seen how the form of products in **S** is determined by their simpler form in **T**. We regard synchronisation trees as transition systems via the inclusion functor, form the limit there and then transport it to **S**, using the fact that right adjoints preserve limits. The coreflection also provides clues as to the form of colimits in **S** because if they exist they will be preserved by the left adjoint (the inclusion functor identifying a synchronisation tree with a transition system). For example, the coproduct of trees is simply their coproduct regarded as transition systems because the coproduct of such transition systems is itself a synchronisation tree. Lemma 83 plays an important role in transporting cartesian and cocartesian morphisms across a coreflection. It is important in showing how restriction and relabelling are preserved. These mathematical facts are important at the level of relating semantics of particular languages in different models.

Imagine giving semantics to the process language **Proc** of section 2.3 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

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

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

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

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 preservation properties adjoints are useful directly in relating different semantics. Less directly, a knowledge of what we can and cannot expect to be preserved 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.

Chapter 3

Noninterleaving models

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 tag, if a 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 certainly not currently 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 an 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.

3.1 Trace languages

3.1.1 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 different understandings of the alphabet; in Mazurkiewicz 's original work the alphabet is thought of as consisting of events, while some authors have instead interpreted its elements as labels, for example standing for port names.

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), 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 a 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 $\simeq$ to be the smallest equivalence relation such that

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

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

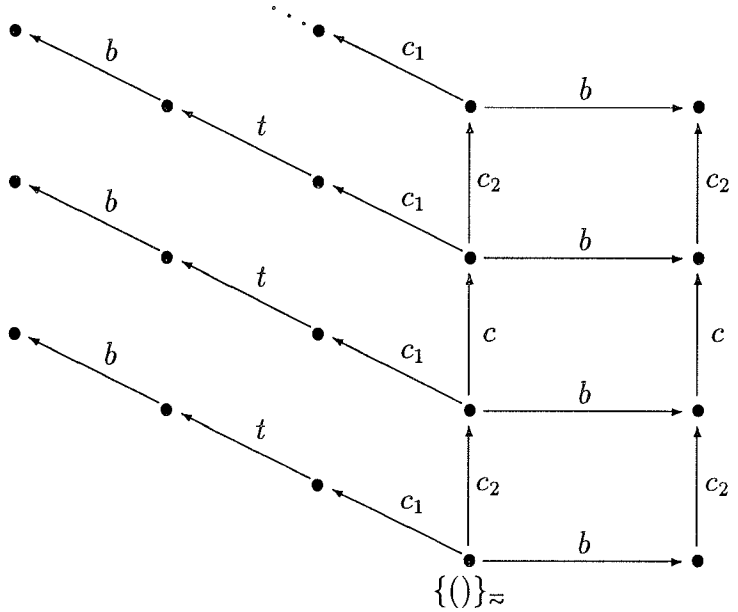
$$s \lesssim t \Leftrightarrow \exists u.\ su \simeq t.$$

Proposition 15 *Let (M, L, I) be a trace language with trace equivalence $\simeq$. If $su \in M$ and $s \simeq s'$ then $s'u \in M$ and $su \simeq s'u$. The relation $\lesssim$ of the trace language is a preorder. Its quotient $\lesssim / \simeq$ by the equivalence relation $\simeq$ is a partial order on traces.*

Example: The independence relation of Mazurkiewicz allows us to express the concurrency we remarked on earlier in example 2.4. 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 2.6 collapses to give the following ordering $\lesssim / \simeq$ 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, the 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 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 16 *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.

3.1.2 Constructions

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 $\pi_0 : L \to_* L_0$ and $\pi_1 : L \to_* L_1$, 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

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

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. The difficulty stems from our regarding the alphabet of a Mazurkiewicz trace language as a set of labels of the kind used in the operations of restriction and relabelling.

Here it is appropriate to discuss an ambiguity in how to apply trace languages to the modelling of parallel computation. In the literature one finds two ways in which to view and use trace languages to 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. 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 algebras. As we have 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 best 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.

3.2 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.

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 $\mathbb{W}$ 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.

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

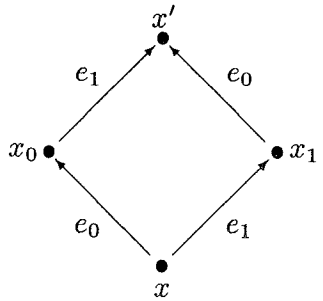
- $e \leq e' \Leftrightarrow \forall x \in \mathcal{D}^0(E, \leq, \#).\ e' \in x \Rightarrow e \in x$.
- $e \# e' \Leftrightarrow \forall x \in \mathcal{D}^0(E, \leq, \#).\ e \in x \Rightarrow e' \notin x$.
- $e \ co \ e' \Leftrightarrow \exists x, x' \in \mathcal{D}^0(E, \leq, \#).\ e \in x \ \& \ e'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

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 18 *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*



3.2.1 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 [38].)

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

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

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

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

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 19 *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 [24].∎

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

Theorem 20 *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

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

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
$\phi(d) = \{p \sqsubseteq d \mid p$ is a complete prime$\}$ with inverse $\theta : \mathcal{D}(P, \leq, \#) \to (D, \sqsubseteq)$ given by $\theta(x) = \bigsqcup x$.

Proof: See [24].∎

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 [5].)

Theorem 21 *The finitary, prime algebraic domains are precisely the dI-domains of Berry.*

Proof: See [38, 33].∎

3.2.2 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 events in ES imply the simultaneous occurrence of events 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 18, showing how the concurrency relation holding between events appears as a "little square" of configurations.

Proposition 22 *Let E be an event structure with concurrency relation co and E' an event structure with concurrency relation co'. Let $\eta : E \to E'$ be a morphism of event structures. Then, for any events e_0, e_1 of E,*

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

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

Proposition 23 *A morphism of event structures from* $(E, \leq, \#)$ *to* $(E', \leq', \#')$ *is a partial function* $\eta : E \rightarrow_* E'$ *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 24 *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*

$$\begin{aligned} e \leq e' \Leftrightarrow & (\exists e_0, e_0'.\ e_0 \leq_0 e_0' \ \&\ j_0(e_0) = e \ \&\ j_0(e_0') = e') \text{ or} \\ & (\exists e_1, e_1'.\ e_1 \leq_1 e_1' \ \&\ j_1(e_1) = e \ \&\ j_1(e_1') = e') \end{aligned}$$

and

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

with injections $j_0 : E_0 \rightarrow E_0 \uplus E_1, j_1 : E_1 \rightarrow E_0 \uplus E_1$ *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 will follow from the coreflection from event structures to trace languages (see corollary 37), and we postpone the construction till then.

3.3 Event structures and trace languages

3.3.1 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

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

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

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

Lemma 25 *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 $\approx$.∎

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 26 *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 27 *Let* $s, t \in M$.

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

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

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

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

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

"⇐": This is proved by induction on s. The basis $s = ()$ 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 26 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 25. Hence $t \approx s'u_0au_1 \approx s'au_0u_1$ making $s'a \lesssim t$. This proves the induction step. ∎

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

Lemma 28 *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 $t = ()$ 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 27, $t' \lesssim u'$ so $t'w \simeq u'$ for some string w. Assume w has the form $b_1 \ldots b_k$. By lemma 27 and proposition 26, we necessarily have $\{t'b_1 \ldots b_i\}_\sim \in ev(s)$ for all i where $0 < i \leq k$. Let

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

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 $u_{i-1} = t'b_1 \ldots b_{i-1}a \in M$. Because $\{t'b_1 \ldots b_{i-1}a\}_\sim \in ev(t)$ and $\{t'b_1 \ldots b_{i-1}b_i\}_\sim \in ev(s)$ by assumption we have $a = b_i$ or aIb_i. However $a \neq b_i$ because otherwise $u_i = t'b_1 \ldots b_{i-1}b_i$ making $\{t'a\}_\sim \in ev(s)$, contrary to our assumption. Now that we know aIb_i the coherence axiom on trace languages ensures

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

In addition

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

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

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

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

$$\begin{aligned} N(a, v) &= N(a, u_0) \quad \text{as } va \sim u_0 a \text{ by lemma 25,} \\ &< N(a, u_0 a u_1). \end{aligned}$$

But $N(a, v) = N(a, u_0 a u_1)$ as $vb \sim u_0 a u_1 b$ by lemma 25. 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. ▮

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 29 *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 27 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 27.

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 $s_1 I b$. Thus $sa = s_0 b s_1 a \sim s_0 s_1 b a \sim s_0 s_1 a$. 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. ∎

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 30 *Let $T = (M, L, I)$ be a trace language. Then the structure $tle(T) = (E, \leq, \#)$ given by definition 3.3.1 is an event structure for which*

$$e \leq e' \quad \text{iff} \quad \forall s \in M.\ e' \in ev(s) \Rightarrow e \in ev(s)$$
$$e \# e' \quad \text{iff} \quad \forall s \in M.\ e \in ev(s) \Rightarrow e' \notin ev(s).$$

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 31 *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 $a \in L$,

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

Proof: Let $s \in M$. By the "only if" direction of lemma 28 it follows that $ev(s)$ is a conflict-free subset of events. By lemma 29, $ev(s)$ is downwards-closed with respect to $\leq$. The fact that Ev is well-defined, 1-1, order preserving and reflecting follows from lemma 27. 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 28 ("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. ∎

The representation theorem for trace languages establishes a connection between trace languages and the *pomset* languages of Pratt [27]. 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 [?], [?] and [?] 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 32 *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 (†).
"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 26) 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 31 there is $v \in M$ such that $ev(v) = x$. Assume $\lambda_T(e) = a$ and $\lambda_T(e') = b$. By (†) 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 (†) 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 25 would imply $N(a, v) = N(a, va)$. But this is clearly absurd, yielding aIb. We have produced the $va, vb \in M$ required.∎

3.3.2 A coreflection

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

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

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

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

Proposition 33 *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 $E \to E'$. However, this follows from the proposition 22 and the observation that if a sequence of events s is associated with a chain of configurations in E then $\hat{\eta}(s)$ is associated with a chain of configurations in E'. ∎

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

Proposition 34 *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 $e_1e_2\cdots e_n$ 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 31, we obtain that $\widehat{\lambda_T}(e_1e_2\cdots e_n) \in M$. If e *co* e', for events e, e', then by proposition 32, 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.∎

Lemma 35 *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* **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 $\hat{\lambda}(se) = \hat{\lambda}(s'e)$. ∎

Theorem 36 *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 35 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 23, 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

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

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 32.
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 30. 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 25. 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 $\eta(e) = \eta'(e)$. ∎

Corollary 37 *The operation tle on trace languages extends to a functor, right adjoint to etl, forming a coreflection between* **E** *and* **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 36 that *tle* extends to a functor, acting as described, so that the pair of functors form an adjunction. From the proof of theorem 36, 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

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

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.

3.4 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 is [1]). Roughly, a Petri net can be thought of as a transition system where, instead of a transition ocurring 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 as Petri net into 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*, and write $e_0 I_N e_1$, iff ${}^\bullet e_0^\bullet \cap {}^\bullet e_0^\bullet = \emptyset$.
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:

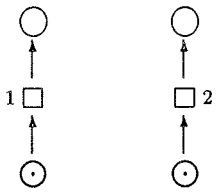
Proposition 38 *Let N be a net with markings M, M' and event e. Then*

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

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 definition 3.4 takes place.

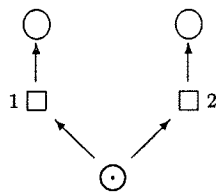
Example:
(1) Concurrency:



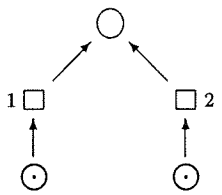
The events 1 and 2 can occur concurrently, in the sense that they both have concession and are independent in not having any pre or post conditions in common.

(2)

Forwards conflict:

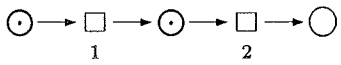


Backwards conflict:



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.

3.4.1 A category of Petri nets

As morphisms on nets we take:

Definition: Let $N = (B, M_0, E, pre, post)$ and $N' = (B', M_0', pre', post')$ 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 $\eta : E \to E'$ and 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 39 *Let $N = (B, M_0, E, pre, post)$, $N' = (B', M_0', E', pre', post')$ be nets. Suppose $(b, \eta) : N \to N'$ is a morphism of net.*

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

Proof: It is easily seen that

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

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. ∎

Proposition 40 *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.

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 $N_1 = (B_1, M_1, L_1, pre_1, post_1)$ be nets. Define their *sum* $N_0 + N_1$ 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

$$\begin{array}{rcl}
b_0 j_0 c & \Leftrightarrow & \exists b_1 \in B_1 \cup \{*\}.\ c = (b_0, b_1) \\
b_1 j_1 c & \Leftrightarrow & \exists b_0 \in B_0 \cup \{*\}.\ c = (b_0, b_1), \qquad \text{and further} \\
M & = & M_0 \times M_1, \\
E & = & E_0 \uplus E_1, \text{ a disjoint union associated with injections} \\
 & & in_0 : L_0 \to L_1, in_1 : L_1 \to L, \text{ and finally} \\
pre(e) & = & j_0 \circ pre_0(e_0) \text{ if } e = in_0(e_0) \text{ and} \\
pre(e) & = & j_1 \circ pre(e_1) \text{ if } e = in_1(e_1).
\end{array}$$

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

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

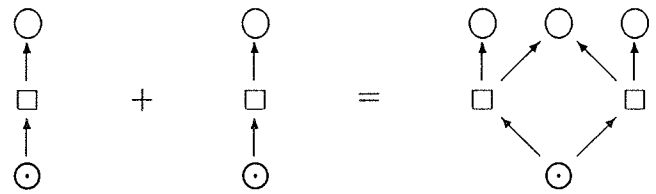
of nets N, N', with conditions B, B' and initial markings M, M' respectively, corresponds to a pair of functions

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

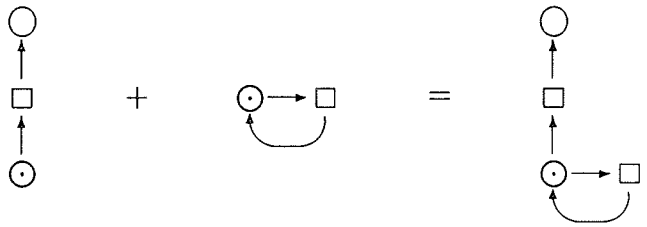
Thus it is to be expected that the conditions of a coproduct of nets correspond to products in $\mathbf{Set}_* \times \mathbf{Set}$. This remark handles the only obstacle in the proof of:

Proposition 41 *The sum* $N_0 + N_1$ *above is a coproduct in the category of nets* **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 sum 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 sum is safe too and has a behaviour which can be described simply in terms of that of the components using the injection morphisms.

Lemma 42 *Suppose N_0, N_1 are safe nets. Then their sum $N_0 + N_1$ is safe. Moreover:*

1. *Two events e, e' are independent in $N_0 + N_1$ iff*
   either
   $$e = in_0(e_0) \;\&\; e' = in_0(e_0') \text{ for events } e_0, e_0' \text{ independent in } N_0$$
   or
   $$e = in_1(e_1) \;\&\; e' = in_1(e_1) \text{ for events } e_1, e_1' \text{ independent in } N_1.$$

2. *X is reachable & $X \xrightarrow{e} X'$ in $N_0 + N_1$ iff*
   either
   $$\exists e_0, X_0, X_0'.\ e = in_0(e_0) \;\&\; X_0 \xrightarrow{e_0} X_0' \;\&\; X = j_0 X_0 \;\&\; X' = j_0 X_0' \text{ in } N_0$$
   or
   $$\exists e_1, X_1, X_1'.\ e = in_1(e_1) \;\&\; X_1 \xrightarrow{e_1} X_1' \;\&\; X = j_1 X_1 \;\&\; X' = j_1 X_1' \text{ in } N_1$$

Proof: (1) is obvious. 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 events 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 $j_0X_0 \xrightarrow{e} X'$ 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_0X_0

$$^\bullet in_1(e_1) \subseteq j_0X_0$$

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_0X_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

$$M_0 \times \{b_1\} \subseteq j_0X_0$$

it follows that $M_0 \subseteq X_0$. But N_0 is safe 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

$$\begin{aligned} X' =& (j_0X_0 \setminus^\bullet in_1(e_1)) \cup in_1(e_1)^\bullet \\ =& (M_0 \times M_1 \setminus^\bullet in_1(e_1)) \cup in_1(e_1)^\bullet \\ =& (j_1M_1 \setminus j_1^\bullet e_1) \cup j_1e_1^\bullet \\ =& j_1((M_1 \setminus^\bullet e_1) \cup e_1^\bullet \\ =& j_1X_1' \end{aligned}$$

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.

$N_0 + N_1$ is readily seen to be a net. Suppose it 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. ∎

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

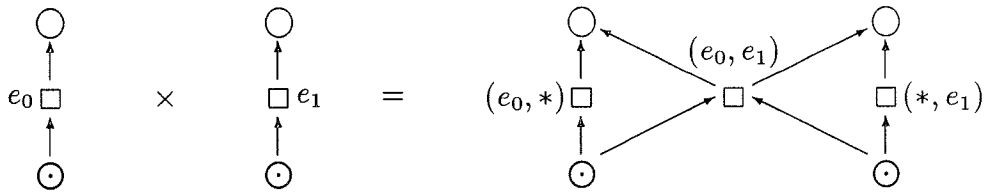
$$pre(e) = \rho_0^{op}[pre_0(\pi_0(e))] + \rho_1^{op}[pre_1(\pi_1(e))]$$
$$post(e) = \rho_0^{op}[post_0(\pi_0(e))] + \rho_1^{op}[post_1(\pi_1(e))].$$

Proposition 43 *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*

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

Proposition 44 *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.*

Example: The product of two nets:



3.5 Asynchronous transition systems

Asynchronous transition systems deserve to be better known as a model of parallel computation. They were introduced independently by Bednarczyk in [4] and Shields in [28]. 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 $(S, i, E, I, Tran)$ 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) $e \in E \Rightarrow \exists s, s' \in S.\ (s, e, s') \in Tran$

(2) $(s, e, s') \in Tran \;\&\; (s, e, s'') \in Tran \Rightarrow s' = s''$

(3) $e_1 I e_2 \;\&\; (s, e_1, s_1) \in Tran \;\&\; (s, e_2, s_2) \in Tran$
$\Rightarrow \exists u.\ (s_1, e_2, u) \in Tran \;\&\; (s_2, e_1, u) \in Tran$

(4) $e_1 I e_2 \;\&\; (s, e_1, s_1) \in Tran \;\&\; (s_1, e_2, u) \in Tran$
$\Rightarrow \exists s_2.\ (s, e_2, s_2) \in Tran \;\&\; (s_2, e_1, u) \in Tran$

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.

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

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

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

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 **A** for the category of asynchronous transition systems.

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

Definition: Assume asynchronous transition systems $T_0 = (S_0, i_0, E_0, I_0, Tran_0)$ and $T_1 = (S_1, i_1, E_1, I_1, Tran_1)$. Their *product* $T_0 \times T_1$ is $(S, i, E, I, Tran)$ where $(S, i, E, Tran)$ is the product of transition systems $(S_0, i_0, E_0, Tran_0)$ and $(S_1, i_1, E_1, Tran_1)$ 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 $T_0 = (S_0, i_0, E_0, I_0, Tran_0)$ and $T_1 = (S_1, i_1, E_1, I_1, Tran_1)$. Their *coproduct* $T_0 + T_1$ is $(S, i, E, I, Tran)$ where $(S, i, E, Tran)$ is the coproduct of transition systems $(S_0, i_0, E_0, Tran_0)$ and $(S_1, i_1, E_1, Tran_1)$ with injections (in_0, j_0) and (in_1, j_1), and the independence relation I is given by

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

3.6 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 $tla(M, E, I) = (S, i, E, I, Tran)$ where

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

Let $\eta : (M, E, I) \to (M', E', I')$ be a morphism of trace languages. Define $tla(\eta) = (\sigma, \eta)$ 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 16.)

Proposition 45 *The operation tla is a functor* $\mathbf{TL}_0 \to \mathbf{A}$.

An asynchronous transition system determines a trace language.

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

$$(s, e_1, s_1), (s_1, e_2, s_2), \ldots, (s_{n-1}, e_n, s_n) \in Tran$$

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

Proposition 46 *The operation atl is a functor* $\mathbf{A} \to \mathbf{TL}_0$.

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

Theorem 47 *The functor tla* : $\mathbf{TL}_0 \to \mathbf{A}$ *is left adjoint to atl* : $\mathbf{A} \to \mathbf{TL}_0$.

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

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

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

$$\eta : L \to atl(T)$$

there is a unique morphism of asynchronous transition systems

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

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

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

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. ▮

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 from event structures **E** to trace languages **TL** restricts to a coreflection to $\mathbf{TL}_0$. 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 48 *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 31, and its consequence, proposition 32, 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 37 ensures that the counit at $etl(T)$

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

is an isomorphism. This makes the function $\lambda_{etl(ES)}$ a bijection, which together with the bijection Ev given by the representation theorem 31, 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. ▮

3.7 Asynchronous transition systems and nets

There is an adjunction between the categories **A** and **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 $na(N) = (S, i, E, I, Tran)$ where

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

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 49 *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 3.5 are obvious while properties (3) and (4) follow directly from the interpretation of independence of events e_1, e_2 as ${}^\bullet e_1 \cap {}^\bullet e_2 = \emptyset$. Letting $(\beta, \eta) : N \to N'$ be a morphism of nets, proposition 39 yields that $na(\beta, \eta)$ is a morphism $na(N) \to na(N')$. Clearly na preserves composition and identities.∎

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

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

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 50 *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 first prove (I) (the proof of (II) is similar):

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

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 3.7:

(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 $T = (S, i, E, I, Tran)$ and $T' = (S', i', E', I', Tran')$. For $b \subseteq Tran'_*$, define

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

Lemma 51 *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,*

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

for any event e of T.

Proof: We show (1), assuming $b \subseteq Tran'_*$ 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 51,

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

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 52 *Let $T = (S, i, E, I, Tran)$ be an asynchronous transition system. Suppose $X \subseteq E$ 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. ∎

Lemma 53 *Let $T = (S, i, E, I, Tran)$ be an asynchronous transition system. Then $an(T)$ is a net. Moreover,*

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

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 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 52 produces a pre and post condition of e.

If $e_1 I e_2$ then axiom (3) in conditions (definition 3.7) ensures ${}^\bullet e_1^\bullet \cap {}^\bullet e_2^\bullet = \emptyset$. Suppose $(s, e, s') \in Tran$. Then, letting B be the set of conditions of T,

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

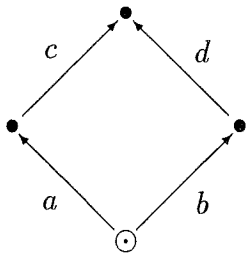
Thus $M(s) \xrightarrow{e} M(s')$. ∎

We illustrate how a net is produced from an asynchronous transition system. The construction produces an awful lot of conditions, so in order to make the drawing of the associated net more managable, we can only draw those conditions which are connected in the sense of the following definition. The transitions and independence of the net are determined by the connected conditions.

Definition: Say a condition b of an asynchronous transition system is *connected* iff there are *not* conditions b_0, b_1 for which

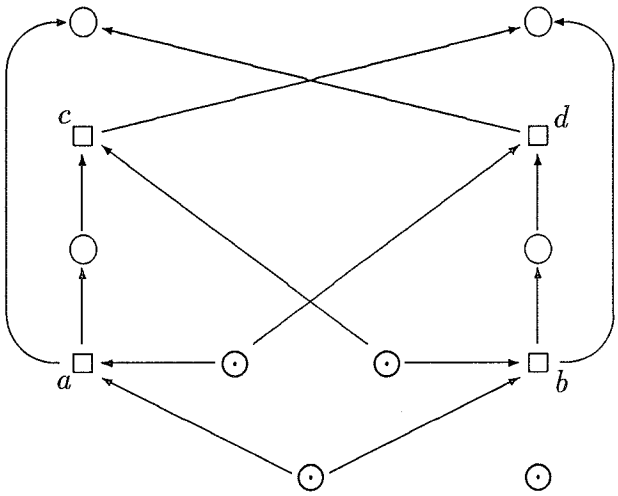
$$b = b_0 \cup b_1 \ \& \ b_0 \cap b_1 = \emptyset.$$

Example: The asynchronous transition system



cId

gives rise, under an, to the following net, where we only draw its connected conditions:



Lemma 54 *an is a functor* $\mathbf{A} \rightarrow \mathbf{N}$.

Proof: The only difficulty comes in showing the well-definedness of an on morphisms. Let $(\sigma, \eta) : T \rightarrow T'$ be a morphism of asynchronous transition systems $T = (S, i, E, I, Tran)$, $T' = (S', i', E', I', Tran')$. We require that $an(\sigma, \eta) =_{def} (\beta, \eta)$ is a morphism of nets $an(T) \rightarrow an(T')$. 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:

$$\begin{aligned} b' \in M_0' &\Leftrightarrow (i', *, i') \in b' \\ &\Leftrightarrow (\sigma(i), *, \sigma(i)) \in b' \\ &\Leftrightarrow (i, *, i) \in (\sigma, \eta)^{-1} b' \\ &\Leftrightarrow \beta^{op}(b') \in M_0. \end{aligned}$$

The fact that $\beta e^\bullet =^\bullet \eta(e)$ and $\beta^\bullet e = \eta(e)^\bullet$ follows directly from (1) and (2) of lemma 51. ∎

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 55 *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 53. ∎

The counit:

Lemma 56 *Let $N = (B, M_0, E, {}^\bullet(\ ), (\ )^\bullet)$ be a net. Let Tran be the transitions of $na(N)$. For $b \in B$ and $c \subseteq Tran_*$, 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 50, $|b|$ 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

$$\begin{aligned} \beta_0^{op}(b) \in M_0' &\Leftrightarrow (M_0, *, M_0) \in \beta_0^{op}(b) \\ &\quad \text{by the definition of } an \text{ and } na, \\ &\Leftrightarrow b \in M_0 \text{ by the definition of } \beta_0. \end{aligned}$$

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 50. ∎

Now we establish the adjunction between **A** and **N** in which an is left adjoint to na.

Lemma 57 *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, σ 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 55, 56. 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 $b \in B$. For 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 (†),

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

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 $\varphi \circ \theta(\beta,\eta) = (\beta,\eta)$. ∎

Theorem 58 *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 55, 56.*

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 55. 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 57, $\theta(h)$ is this composition. ∎

3.7.1 A coreflection

Neither **A** nor **N** embeds fully and faithfully in the other category. This accompanies the facts that neither unit nor counit is an isomorphism (see [18] 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 transitions

$$(i, e_1, s_1), (s_1, e_2, s_2), \ldots, (s_{n-1}, e_n, s) \in Tran.$$

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 one of 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 here:

Proposition 59 *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 transition of T. It has the following properties:

$$\begin{aligned} (s, *, s) \in \overline{b} &\Leftrightarrow (s, *, s) \notin b, \text{ for any } s \in S, \\ \overline{b} \in^\bullet e &\Leftrightarrow b \in e^\bullet \;\&\; b \notin^\bullet e \\ \overline{b} \in e^\bullet &\Leftrightarrow b \in^\bullet e \;\&\; b \notin e^\bullet \text{ for any } e \in E. \end{aligned}$$

Let $(\sigma, \eta) : T' \to T$ be a morphism of asynchronous transition systems and b be a condition of T. Then

$$(\sigma, \eta)^{-1}\overline{b} = \overline{(\sigma, \eta)^{-1}b}.$$

Proof: Let b be a condition of T. We note the following facts:

$$\begin{aligned} (s, e, s') \in^\bullet \overline{b} &\Leftrightarrow (s, *, s) \in b \;\&\; (s', *, s') \notin b \\ &\Leftrightarrow (s, e, s') \in b^\bullet \;\&\; (s', *, s') \notin^\bullet b \\ (s, e, s') \in \overline{b}^\bullet &\Leftrightarrow (s, *, s) \notin b \;\&\; (s', *, s') \in b \\ &\Leftrightarrow (s, e, s') \notin b^\bullet \;\&\; (s, e, s') \in^\bullet b \end{aligned}$$

for any transition (s, e, s') of T.

We now show $\overline{b}$ is a condition of T:

(1) Suppose $(s, e, s') \in \overline{b}$. Then directly from the definition of b we see $(s, *, s), (s', *, s') \in b$.

(2) (i) Thus if (s, e, s') and (u, e, u') are both transitions of T, because b is a condition, we obtain using the facts above, that

$$\begin{aligned} (s, e, s') \in^\bullet \overline{b} &\Leftrightarrow (s, e, s') \in b^\bullet \;\&\; (s, e, s') \notin^\bullet b \\ &\Leftrightarrow (u, e, u') \in b^\bullet \;\&\; (u, e, u') \notin^\bullet b \\ &\Leftrightarrow (u, e, u') \in^\bullet \overline{b}. \end{aligned}$$

The proof of (2)(ii) is similar.

(3) If $(s, e_1, s') \in^\bullet \overline{b}^\bullet \;\&\; (u, e_2, u') \in^\bullet \overline{b}^\bullet$ then, by the facts above $(s, e_1, s') \in {}^\bullet b^\bullet \;\&\; (u, e_2, u') \in {}^\bullet b^\bullet$. As b is a condition, $\neg e_1 I e_2$.

The stated properties of $\overline{b}$ are obvious from its definition and the facts above.

Letting $(\sigma, \eta) : T' \to T$ be a morphism of asynchronous transition systems and b be a condition of T, we see

$$\begin{aligned} (s, e, s') \in (\sigma, \eta)^{-1}\overline{b} &\Leftrightarrow (\sigma(s), \eta(e), \sigma(s')) \notin b \;\&\; (\sigma(s), *, \sigma(s)) \notin b \;\&\; (\sigma(s'), *, \sigma(s')) \notin b \\ &\Leftrightarrow (s, e, s') \in \overline{(\sigma, \eta)^{-1}b}. \blacksquare \end{aligned}$$

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, $|b| \notin M(u)$ and $|b| \in M(s)$. With complementation we can separate the other way:

$$\overline{|b|} \in M(u), \quad \overline{|b|} \notin M(s)$$

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 60 *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 a 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 $s \overset{e}{\not\rightarrow}$ 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

$$|b| \in e^\bullet\ \&\ |b| \in M(s)\ \&\ |b| \notin e^\bullet.$$

But then, taking the complement of $|b|$,

$$\overline{|b|} \in^\bullet e\ \&\ \overline{|b|} \notin M(s),$$

by proposition 59.

In either case, (i) or (ii), we obtain a condition c of $na(N)$ for which

$$c \in^\bullet e\ \&\ c \notin M(s).\blacksquare$$

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 safe. Consequently $an \circ na(N)$ yields a safe net for any net N.

Proposition 61 *Assume $T = (S, i, E, I, Tran)$ is an asynchronous transition system satisfying Axioms 2 and 3 above. Then*

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

for any $s, s' \in S$ and $e \in E$. In addition, $an(T)$ is a safe net.

Proof: By lemma 53,

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

To establish the converse, with the assumption of Axioms 2 and 3, we first observe that

$$^\bullet e \subseteq M(s) \Rightarrow \exists s_1.\ (s, e, s_1) \in Tran$$

for $s \in S$ and $e \in E$. Otherwise as there is some $(u, e, u') \in Tran$ there would be a contradiction of Axiom 3. Now, suppose $M(s) \xrightarrow{e} M(s')$. Then $^\bullet e \subseteq M(s)$ so $(s, e, s_1) \in Tran$ from some state s_1. Thus $M(s) \xrightarrow{e} M(s_1)$ and now by Axiom 2 we deduce $s' = s_1$, and hence the converse $M(s) \xrightarrow{e} M(s') \Rightarrow (s, e, s') \in Tran$.

It follows that any reachable marking of $an(T)$ has the form $M(s)$ for some $s \in S$. Assuming $^\bullet e \subseteq M(s)$ there is as above, $M(s) \xrightarrow{e} M(s')$ for some transition (s, e, s') of T. 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 therefore safe. ∎

Corollary 62 *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 $T = (S, i, E, I, Tran)$ as follows:

$$\mathcal{R}(T) = (S', i', E', I', Tran')$$

where

$$\begin{aligned} S' &\quad \text{consists of all reachable states of } T \\ E' &= \{e \in E \mid \exists s, s' \in S'.\ (s, e, s') \in Tran\} \\ I' &= I \cap (E' \times E') \\ Tran' &= Tran \cap (S' \times E' \times S'). \end{aligned}$$

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 63 *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. ▮

Proposition 64 *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. ▮

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 65 *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

$$c \beta b \text{ iff } \emptyset \neq c = |b|^R$$

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 $T = (S, i, E, I, Tran) \in \mathbf{A}_0$ a morphism in $\mathbf{A}_0$

$$(\sigma, 1_E) : T \to na_0 \circ an(T)$$

where $\sigma(s) = M(s)$ for $s \in S$. It remains to argue why $(\sigma, 1_E) : T \to na_0 \circ an(T)$ is an isomorphism.

States of $na_0 \circ an(T)$ are reachable markings and its transitions are transitions of the net $an(T)$ from reachable markings. The independence relation for $na \circ an(T)$ holds between events e_1, e_2 iff ${}^\bullet e_1^\bullet \cap {}^\bullet e_1^\bullet = \emptyset$. From proposition 61, as T satisfies Axioms 2 and 3,

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

It follows that all reachable markings of $an(T)$ have the form $M(s)$ for some $s \in S$. Because T satisfies Axiom 2, $s \mapsto M(s)$ is thus a bijection between S and states of $na \circ an(T)$. Thus for $(\sigma, 1_E)$ to be an isomorphism it suffices now to show

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

for $e_1, e_2 \in E$. The "$\Rightarrow$" direction of the equivalence follows directly from $(\sigma, 1_E)$ being a morphism. To show the converse, "$\Leftarrow$" direction, suppose $\neg e_1 I e_2$ and apply lemma 52 to $\{e_1, e_2\}$ to obtain a condition in ${}^\bullet e_1^\bullet \cap {}^\bullet e_1^\bullet$, which must therefore be non-empty. It follows that $(\sigma, 1_E) : T \cong na_0 \circ an(T)$. 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 64 and lemma 57.∎

One consequence of the coreflection is that $\mathbf{A}_0$ has products and coproducts given by the same constructions as those of $\mathbf{A}$.

Proposition 66 *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 43, 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 42 their coproduct in $\mathbf{A}$ is isomorphic to $na_0(an(T_0) + an(T_1))$.∎

The coreflection $an_0 : \mathbf{A}_0 \to \mathbf{N}$, $na_0 : \mathbf{N} \to \mathbf{A}_0$ 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 67 *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 65: $na_0 = \mathcal{R} \circ na : \mathbf{N} \to \mathbf{A}_0$ 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 if $N \in \mathbf{N}_0$. 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 68 *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 $\mathbf{E}$ to asynchronous transition systems $\mathbf{A}$. In fact the coreflection cuts down to one between $\mathbf{E}$ and $\mathbf{A}_0$.

Proposition 69 *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 48. 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 [24])(see *e.g.* the construction of an occurrence net from an event structure in [24]). Hence by theorem 65, $na_0(N)$, and so the isomorphic $tla \circ etl(E)$, belong as objects to $\mathbf{A}_0$.∎

3.8 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 2.3. 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 which is 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 2.3, which generate the transitions, to include extra rules which express the independence between transitions.

3.8.1 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 [3]. 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

$$I \xrightarrow{!} F(I) \overset{F(!)}{\to} F^2(I) \overset{F^2(!)}{\to} \cdots \overset{F^{(n-1)}(!)}{\to} F^n(I) \overset{F^n(!)}{\to} \cdots$$

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 eachother. In the case of event structures the embeddings have already been introduced and studied by Kahn and Plotkin under the name *rigid embeddings* (see [14, 37]).

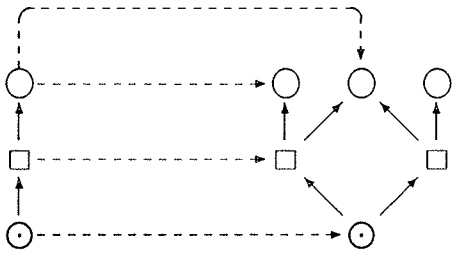
Petri nets: We first consider *embeddings* between nets. These are simply monomorphisms in the category $\mathbf{N}$.

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 70 *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*

$$b \in pre(e) \Leftrightarrow \forall k \in \omega.\ (e_k \neq * \Rightarrow (b_k \neq * \ \&\ b_k \in pre_k(e_k)))$$
$$b \in post(e) \Leftrightarrow \forall k \in \omega.\ (e_k \neq * \Rightarrow (b_k \neq * \ \&\ b_k \in post_k(e_k))),$$

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 71
(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 *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

$M \subseteq M'$
$E \subseteq E'$ and
$aIb \Leftrightarrow aIb, \quad$ for all $a, b \in E$.

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 [14, 37]). 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 $ES_0 = (E_0, \leq_0, \#_0), ES_1 = (E_1, \leq_1, \#)$ be event structures. Define $ES_0 \trianglelefteq ES_1$ iff

$$E_0 \subseteq E_1,$$

$$\forall e \in E_1 . e \leq_0 e_0 \Leftrightarrow e \leq e_1,$$

for all $e_0 \in E_0$, and

$$e \#_0 e' \text{ iff } e \#_1 e_1,$$

for all $e, e' \in E_0$.

The $\trianglelefteq$ order on event structures is a special case of the order on trace languages:

Proposition 72

(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.

3.8.2 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. 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 sets of events 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 **Set**,
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 **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 73 *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 74 *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

$$e \# e' \Leftrightarrow e, e' \in E_0 \ \&\ e \#' e'.$$

Generally, for the other models as well, restriction with respect to a subset of events can be expressed as strong cartesian lifting. Let $\mathbf{X}$ be a category of structures, like event structures, nets or asynchronous transition systems. Let $E_X : \mathbf{X} \to \mathbf{Set}_*$ be the functor yielding the event sets. Define a new category, called $(\mathbf{X}, E_X)$ consisting of

objects (X, E) where $E_X(C) \subseteq E$,

morphisms $(f, \eta) : (X, E) \to (X', E')$ where $f : X \to X'$ is a morphism in $\mathbf{X}$ and $\eta : E \to_* E'$ is a morphism in $\mathbf{Set}_*$ such that

$$\begin{array}{ccc} E_X(X) & \hookrightarrow & E \\ E_X(f) \downarrow & & \downarrow_* \eta \\ E_X(X') & \hookrightarrow & E' \end{array}$$

commutes. The composition $(f, \eta) \circ (f', \eta')$ is defined to be $(f \circ f', \eta \circ \eta')$.

There is a clearly a functor $q : (\mathbf{X}, E_X) \to \mathbf{Set}_*$ taking a morphism $(f, \eta) : (X, E) \to (X', E')$ to $\eta : E \to E'$. The categories we consider have operations of restriction on events characterised as strong cartesian liftings of inclusions $E \hookrightarrow E'$ with respect to the functor q and any object (X', E') in $(\mathbf{X}, E_X)$. For such categories $\mathbf{X}$, there is an operation of restriction determined by subsets of labels on the labelled versions $\mathcal{L}(\mathbf{X})$. Recall the functor $p : \mathcal{L}(\mathbf{X}) \to \mathbf{Set}_*$; a morphism of labelled structures

$$(f, \lambda) : (X, l : E_X(X) \to L) \to (X', l' : E_X(X') \to L')$$

is sent to

$$\lambda : L \to_* L'.$$

For any inclusion $\lambda : L \hookrightarrow L'$ in $\mathbf{Set}_*$, the functor p has a strong cartesian lifting of λ with respect to any object $(X', l' : E_X(X') \to L')$ in $\mathcal{L}(\mathbf{X})$: it is given by the morphism

$$(r, \lambda) : (X, l : E_X(X) \to L) \to (X', l' : E_X(X') \to L')$$

in $\mathcal{L}(\mathbf{X})$ where (r, η) is the strong cartesian lifting of the inclusion on event sets

$$\eta : \{e \in E_X(X') \mid l'(e) \in L\} \hookrightarrow E_X(X')$$

with respect to $q : (\mathbf{X}, E_X) \to \mathbf{Set}_*$, and l is the restriction of l' to the events $E_X(X)$ of X. This yields a suitable restriction operation on $\mathcal{L}(\mathbf{X})$ when $\mathbf{X}$ is specialised to any one of the models event structures, trace languages, Petri nets or asynchronous transition systems.

To treat recursion on labelled structures we extend embeddings (and the special case when they are inclusions) 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 union $\bigcup_{n\in\omega} l_n$ has domain $\bigcup_{n\in\omega} E_n$ and codomain $\bigcup_{n\in\omega} L_n$.

In section 2.3 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, a little 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, it can be checked that, 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 2.3.

3.8.3 Operational semantics

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 2.3.

Definition: A *transition system with independence* 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)

(i) $(s, a, s_1) \prec (s_2, a, u) I (w, b, w') \Rightarrow (s, a, s_1) I (w, b, w')$

(ii) $(w, b, w') I (s, a, s_1) \prec (s_2, a, u) \Rightarrow (w, b, w') I (s_2, a, u)$

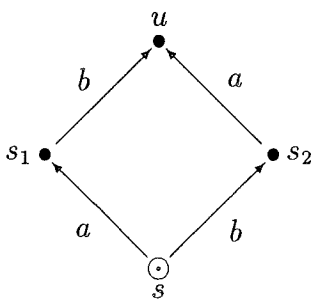
where the relation $\prec$ between transitions is defined by

$$(s, a, s_1) \prec (s_2, a, u) \Leftrightarrow (s, a, s_1) I (s, b, s_2)\ \&\ (s, a, s_1) I (s_1, b, u)\ \&\ (s, b, s_2) I (s_2, a, u),$$

and $\sim$ is the least equivalence relation including $\prec$.

As morphisms on transition systems in with independence we take morphisms on the underlying transition systems which preserve independence; 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 $(s,a,s_1) \prec (s_2,a,u) \Leftrightarrow (s,a,s_1)I(s,b,s_2)$ & $(s,a,s_1)I(s_1,b,u)$ & $(s,b,s_2)I(s_2,a,u)$, 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

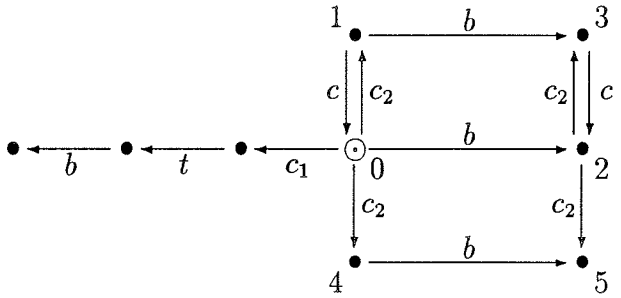
(1) for any label there is at most one event with that label involved in a transition between two states.

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 2.4. 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 2.4,



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.

Operational rules

Transition systems with independence have the striking advantage 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 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:

$$\begin{aligned}
&(s, a, s')I_*(u, b, u') \text{ iff} \\
&(\rho_0(s), \pi_0(a), \rho_0(s'))I_{0*}(\rho_0(u), \pi_0(b), \rho_0(u')) \ \& \\
&(\rho_1(s), \pi_1(a), \rho_1(s'))I_{1*}(\rho_1(u), \pi_1(b), \rho_1(u')).
\end{aligned}$$

We have already seen rules to give the transitions of the product (section 2.3). 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

$(s, a, s')I(u, b, u')$ iff
$[\exists s_0, s_0', u_0, u_0'.$
$s = in_0(s_0)$ & $s' = in_0(s_0')$ & $u = in_0(u_0)$ & $u' = in_0(u_0')$ & $(s_0, a, s_0')I_0(u_0, b, u_0')]$ or
$[\exists s_1, s_1', u_1, u_1'.$
$s = in_1(s_1)$ & $s' = in_1(s_1')$ & $u = in_1(u_1)$ & $u' = in_1(u_1')$ & $(s_1, a, s_1')I_1(u_1, b, u_1')]$.

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 $T = (S, i, L, Tran, I)$ be a transition system with independence, the restriction to a subset of labels Λ is

$$T \restriction \Lambda = (S, i, L', Tran', I')$$

where $L' = L \cap \Lambda$, $Tran' = Tran \cap S \times L' \times S$ and $I' = I \cap Tran \times Tran$. 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 2.3. Then $a = b$.

This property is easily observed to be preserved by the rules.

Let $T = (S, i, L, Tran, I)$ be a transition system with independence, assumed to satisfy the unicity property

$$(s, a, s') \in Tran \;\&\; (s, b, s') \in Tran \Rightarrow a = b.$$

For $\Xi : L \to L'$ the relabelling

$$T\{\Xi\} = (S, i, L', Tran', I'),$$

where $Tran' = \{(s, b, s') \mid \exists a.\ b = \Xi(a) \;\&\; (s, a, s') \in Tran\}$ 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 2.3 for the transitions between (tagged) states by the following rules for the independence relation (also relating idle transitions):

Rules for independence

$$(s,a,s')I(u,*,u) \qquad \frac{(s,a,s')I(u,b,u')}{(u,b,u')I(s,a,s')}$$

$$\frac{(s,a,s')I(t,b,t')}{((n,s),a,(n,s'))I((n,t),b,(n,t'))}$$

Sum:

$$\frac{(s,a,s')I(s,b,s'')}{(s\oplus t,a,(0,s'))I(s\oplus t,b,(0,s''))}\ a\neq *, b\neq * \qquad \frac{(t,a,t')I(t,b,t'')}{(s\oplus t,a,(1,t'))I(s\oplus t,b,(1,t''))}\ a\neq *, b\neq *$$

$$\frac{(s,a,s')I(u,b,u')}{(s\oplus t,a,s')I((0,u),b,(0,u'))}\ u\not\equiv s \qquad \frac{(t,a,t')I(u,b,u')}{(s\oplus t,a,t')I((1,u),b,(1,u'))}\ u\not\equiv t$$

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:

$$\frac{(s,a,s')I(t,b,t')}{(s\restriction\Lambda,a,s'\restriction\Lambda)I(t\restriction\Lambda,b,t'\restriction\Lambda)}a,b\in\Lambda \qquad \frac{(s,a,s')I(t,b,t')}{(s\{\Xi\},\Xi(a),s'\{\Xi\})I(t\{\Xi\},\Xi(b),t'\{\Xi\})}$$

Recursion:

$$\frac{(t[rec\ x.t/x],a,s)I(t[rec\ x.t/x],b,u)}{(rec\ x.t,a,(2,s))I(rec\ x.t,b,(2,u))} \qquad \frac{(t[rec\ x.t/x],a,s)I(u,b,u')}{(rec\ x.t,a,(2,s))I((2,u),b,(2,u'))}\ u\not\equiv t[rec\ x.t/x]$$

A 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 prefixed 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 systems reachable from $rec\ x.t$ and its unfolding $t[rec\ x.t/x]$ (this fact is used

in the proof of theorem 11). The isomorphism is given by

$$\begin{array}{lcl} rec\ x.t & \mapsto & t[rec\ x.t/x] \\ (2,u) & \mapsto & u. \end{array}$$

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 2.3 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. Firstly, the following uniqueness lemma is shown:

Definition: For $T = (S, i, L, tran, I)$ a transition system with independence, define $\mathcal{R}(T)$ to be $(S', i, L', Tran', I')$ 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 $Tran'$ and independence relation $I' = I \cap (Tran' \times Tran')$.

Lemma 75 *If x is guarded in t then*

$$T \cong \mathcal{R}(\mathbf{TI}[\![t]\!]\rho[T/x] \Rightarrow T \cong \mathcal{R}(\mathbf{TI}[\![rec\ x.t]\!]\rho),$$

for any environment ρ.

The proof of this fact is quite involved, and the details are shown in Appendix C (b). The second stage is to show:

Theorem 76 *Let t be a closed process term. Then*

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

for an arbitrary environment ρ.

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 $\overline{x}$,

$$\mathcal{O}p(t[\overline{s/x}]) \cong \mathcal{R}(\mathbf{TI}[\![t]\!]\rho[\overline{\mathcal{O}p(s)/x}]).$$

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 75 above.∎

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 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 $T = (S, i, E, I, Tran, l : E \to L)$ be an object of $\mathcal{L}(\mathbf{A})$. Define $u(T)$ to be $(S, i, L, Tran', I')$ where

$$(s, a, s') \in Tran' \Leftrightarrow \exists e.\ l(e) = a \ \&\ (s, e, s') \in Tran$$
$$(s, a, s')I'(t, b, t') \Leftrightarrow \exists e_0, e_1.\ l(e_0) = a \ \&\ l(e_1) = b \ \&\ (s, e_0, s')I(t, e_1, t').$$

Theorem 77 *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,

$$fixF = u(fixG).$$∎

As we will see, the coreflections between categories of unlabelled structures extend to categories of labelled structures. In particular, this yields a coreflection between $\mathcal{L}(\mathbf{E})$ and $\mathcal{L}(\mathbf{A})$. Because this coreflection cuts down to one between $\mathcal{L}(\mathbf{E})$ and **TI** it follows that they induce the same semantics in labelled event structures.

Open problem: I suspect but don't yet have a proof that the semantics in **TI** is equivalent to one in terms of labelled asynchronous transition systems in $\mathbf{A}_0$, and thus one whose behaviour is induced by a net. Certainly the operation u does not stay within $\mathbf{A}_0$ in general (here **TI** is identified with its equivalent subcategory in $\mathcal{L}(\mathbf{A})$).

3.9 Relating models

Earlier in section 3.8.2, 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.

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 morphisms in $\mathbf{Set}_{*0}$. Define the functor $\mathcal{L}(F, \phi) : \mathcal{L}(\mathbf{C}) \to \mathcal{L}(\mathbf{D})$ to act on objects so

$$(C, l : E_C(C) \to B) \mapsto (F(C), l \circ \phi_C : E_D \circ F(C) \to B)$$

and on morphisms so

$$(f, \lambda) \mapsto (F(f), \lambda)$$

where $(f, \lambda) : (C, l : E_C(C) \to B) \to (C', l' : E_C(C') \to B')$.

Under reasonable conditions the labelling operation $\mathcal{L}(-)$ preserves adjunctions, coreflections and reflections:

Lemma 78 *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 [18] 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.∎

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 $\mathbf{Set}_*$, 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 78. 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 78 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

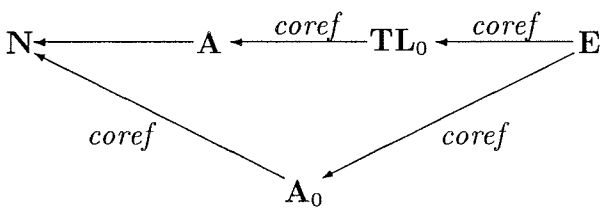
$$\epsilon_{etl(E)} \circ \eta_E = 1_E,$$

i.e. that the first half of (1) holds. The remaining half of (1) reduces to an obvious equality. We conclude by lemma 78 that

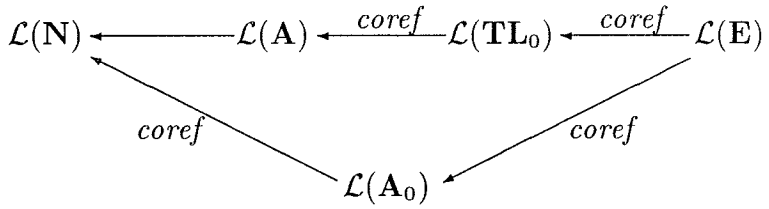
$$\mathcal{L}(etl, 1) : \mathcal{L}(\mathbf{E}) \to \mathcal{L}(\mathbf{TL})$$

forms a coreflection, with right adjoint $\mathcal{L}(tle, \gamma)$. The coreflection $\mathbf{E} \to \mathbf{TL}$ cuts down to one $\mathbf{E} \to \mathbf{TL}_0$, 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 adjunctions and coreflections



to the categories of labelled structures. Lemma 78 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 78 a triviality. One exception is right adjoint of the coreflection from $\mathbf{TL}_0$ to $\mathbf{E}$, dealt with in section 3.8.2. 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 78 lifts the coreflection between asynchronous transition systems and nets to labelled structures. We obtain:

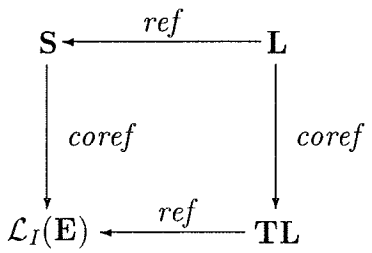


Thus we can use the preservation properties of adjoints to relate constructions, and thus semantics, across different categories of labelled structures.

It remains to relate the interleaving and noninterleaving models of concurrency. Recall we have:

$$\mathbf{T} \overset{coref}{\longleftarrow} \mathbf{S} \overset{ref}{\longleftarrow} \mathbf{L}$$

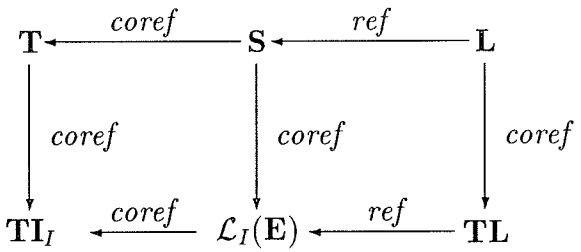
In particular, when we move to the models with explicit independence, what is the analogue of the reflection between languages and synchronisation trees? One analogue is a reflection between Mazurkiewicz trace languages and labelled event structures. In this case the alphabet of a trace language is to be thought of as consisting of labels (not events), and to achieve the adjunction the event structures are labelled by sets with an independence relation. More precisely, the labelling function sends concurrent events to independent labels, and independence is respected by the morphisms, in forming the category $\mathcal{L}_I(\mathbf{E})$.



The right adjoint of the reflection, regarding a trace language as a labelled event structure, is provided by the representation theorem, theorem 31. The two left adjoints identify a synchronisation tree with a labelled event structure, and a language with a trace language; in both cases the original labelling set becomes a set with empty independence relation. Details can be found in [40], an expanded version this account. Because objects in **TL** possess an independence determined by the labels they are not so appropriate as

denotations of processes like those described by the language **Proc**. As we have seen, trace languages can be understood as very special pomset languages. A similar result, but covering a wider class of pomset languages, appears in [30].

There are several candidates for the generalisation of transition systems: labelled Petri nets, labelled asynchronous transition systems, transition systems with independence. However there are not coreflections from transition systems **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 from being a left adjoint. One way to repair this is to extend **T** to allow more than one transition with the same label between two states; there are then coreflections to labelled Petri nets, labelled asynchronous transition systems, labelled Petri nets, labelled asynchronous transition systems. Another way is to insist that labelled asynchronous transition systems be extensional, by replacing them, for instance, by a suitable category of transition systems with independence—see section 3.8.3. A pleasing picture extending the reflections is obtained by choosing transition systems with independence in which the labelling sets also carry an independence relation respected by the morphisms. The category $\mathbf{TI}_I$ consists of objects $(S, i, L, \mathit{Tran}, I_T, I)$ where $(S, i, L, \mathit{Tran}, I_T)$ is a transition system with independence and I is an independence relation on L, with the property that if two transitions are independent according to I_T then then their labels are also independent according to I. An event structure in $\mathcal{L}_I(\mathbf{E})$ can be identified with such a transition system; its transitions are $x \xrightarrow{a} x'$ where $x \xrightarrow{e} x'$, for a pair of configurations x, x' and an event e labelled by a. (More details are to be found in [40].)



Chapter 4

Notes

Labelled transition systems were studied early by Keller in [?]. The use of labelled transition systems is central to the work on CCS [22], processes of which are first given a semantics as labelled transition systems on which equivalences, like bisimulation, are then defined. The labelled transition systems are defined in a syntax directed way using Plotkin's structural operational semantics [25]. CSP and its theoretical variant TCSP have most commonly been attributed with a failure-set semantics. However this can be regarded as an equivalence on a more basic labelled-transition-system semantics (as has been done, for example, in [6, ?]). They can also be used in the semantics of languages with value-passing such as OCCAM (see *e.g.* [?]). The "partial simulation" or "refinement" morphisms we define on transition systems seem to have been discovered several times. Their relevance to languages like CCS and CSP was first pointed out in [34]. Here 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 of the initial states holds, though it is not determined which (a notable difference is that then the coproduct amounts to just disjoint juxtaposition).

Synchronisation algebras were used largely for the purpose of generality in [32]. They can be regarded as generalising Milner's monoids of actions [?] 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 Bergstra and Klop [?].

Synchronisation trees appeared early on in the work of CCS [21]. 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 [10] though the idea did not originate there, for example appearing in the early work on path expressions [17].

One omission from our categorical explication of models 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 $\lambda_!$; 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.

An early reference for Mazurkiewicz traces is [?], though the material can also be found in [20]. Mazurkiewicz traces are generally defined a little differently. In particular it is not usual to insist on the *coherence* axiom in their definition. As remarked, Mazurkiewicz traces correspond to labelled partial orders of events, christened *pomsets* (partially ordered multisets) by Pratt (though note that far from all pomsets can be obtained in this way). Consequently Mazurkiewicz trace languages correspond to special kinds of pomset languages (see [?], [?] for a characterisation).

Event structures were introduced in [24] and their theory developed, including the modelling of higher types in [31] (representing Berry's *bidomains* [5]). Event structures can have a general, and not just a binary conflict, so they represent precisely the dI-domains of Berry (not just the coherent ones)—see *e.g.* [37]. Event structures bear the same relation to dI-domains as do information systems to Scott domains. The characterisations of the domains of configurations as prime algebraic appear in [24] and [31], and the realisation that prime algebraicity amounts to precisely distributivity in [38], [33]. 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 [37, 38]. Provided the more general event structures have dI-domains as domains of configurations an event structure of the form $(E, \leq, \#)$ can always be extracted. This line has been followed in [32, 37, ?]. 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. The first event-structure semantics of TCSP appears in the masters' thesis of Fogh [?], under supervision of Nielsen. Event-structure semantics for CCS/TCSP-like languages was made systematic in [32], which exploited a new definition of morphism—that which appears here. A variation on the idea appears in the "flow event structures" of Boudol and Castellani [?]; however in order that parallel compositions can be defined correctly within them requires an extra axiom [?], the preservation of which necessitates an unusual treatment of restriction, one where the events to be restricted away are made self-conflicting instead of removed.

The relationship between event structures and Mazurkiewicz trace languages seems first to have been made explicit in [?]. However, the proof of the representation theorem here appears to be new. The coreflection from event structures to trace languages is shown in [4].

The study of pomsets as a model of computation has been most famously advocated by Pratt in a series of papers beginning with [27]. Labelled partial orders of events appeared earlier in the study of concurrency (*e.g.* [?], [?]). The relation between pomsets and Mazurkiewicz traces is studied in *e.g.* [?] and [?]. More on the categorical relationship of pomsets to the models here can be found in [30].

A good reference on Petri nets is [1]. The version of Petri nets we describe can be found in the paper [20] of Mazurkiewicz. They are more general than condition-event systems 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 interesting that this construction comes out of the adjunction between nets and asynchronous transition systems. There are several versions of morphism on nets in the literature, some more deserving of attention than others. The original definition by Petri [?] seems to have been motivated by graph-

theoretic considerations—Petri's morphisms do not respect the behaviour of nets. One limitation of the morphisms of nets used here is that they do not allow all of the folding maps (*e.g.* from an unfolding to the original net) one might like. A solution is to take morphisms in which the relation between conditions is more general than the opposite of a partial function. It appears, though has not yet been checked, that the coreflection here generalises to safe nets when β is a relation [37, 36]—in addition to the reachability and separation axioms asynchronous transition systems are further required to satisfy

$$e_1 I e_2 \Rightarrow \exists s, s_1, s_2.\ (s, e_1, s_1), (s, e_2, s_2) \in tran$$

(independence means concurrency), while in forming the left adjoint, conditions of an asynchronous transition are taken to be connected subsets of transitions (with idling transitions). To some extent the ideas presented here generalise to nets in which events can fire and markings hold with multiplicities [39], though at present it is not known how to link up with other models via adjunctions. Recently 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 [?]).

Asynchronous transition systems are due to Bednarczyk [4] and Shields [28] who discovered them independently. Bednarczyk's thesis [4] contains the definition of the category of asynchronous transition systems and the coreflections with event structures and Mazurkiewicz traces. As has been pointed out when presented as transition systems with independence, they are amenable to the same techniques (*e.g.* definition by structural operational semantics) as ordinary transition systems. Alternatively, asynchronous transition systems can be 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 [?, 19] for two 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 $\mathbf{A}_0$. They are a special case of the "geometric" transition systems proposed by Pratt in [?]. The adjunction between asynchronous transition systems and nets is new. It can be viewed as a extension of the adjunction between elementary nets and elementary transition systems in [?].

Acknowledgements

We are grateful to P.S.Thiagarajan, Bart Jacobs, Peter Knijnenburg and Vladimiro Sassone for helpful suggestions. Thanks to Madhaven Mukund for his preparation of the diagrams.

Appendix A

Fibred categories

Our presentation relies on some basic notions from fibred category theory originating in the work of Grothendieck [?], and Bénabou [2].

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 $p(h) = 1_L$ and $f \circ h = g$.
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

- 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.

The fibrations and cofibrations we consider come equipped with a particular choice of cartesian and cocartesian liftings. In more detail the fibrations $p : \mathbf{X} \to \mathbf{B}$ will be such that, for any $\lambda : B \to B'$ in $\mathbf{B}$ and $X' \in p^{-1}(B')$ there will be defined a specific construction $\lambda^*(X')$ and cartesian lifting $c(\lambda, X') : \lambda^*(X') \to X'$ of λ. Such a function c making a choice of cartesian liftings for a fibration is called a *cleavage*. The fibration ensures that two cleavages are the same to within vertical isomorphism; if $c(\lambda, X')$ and $c'(\lambda, X')$ are two choices for the cartesian lifting of λ with respect to X' then there is a unique vertical isomorphism θ such that $c'(\lambda, X') = c(\lambda, X') \circ \theta$. A cleavage for a fibration specifies a functor between fibres extending the construction λ^* on objects. The functor $\lambda^* : p^{-1}(B') \to p^{-1}(B)$ for each morphism $\lambda : B \to B'$ is defined as follows:
Let $g' : X'_0 \to X'_1$ be a morphism in the fibre $p^{-1}(B')$. The cleavage specifies cartesian morphisms

$$c(\lambda, X'_0) : \lambda^*(X'_0) \to X'_0 \text{ and } c(\lambda, X'_1) : \lambda^*(X'_1) \to X'_1.$$

As the morphism $c(\lambda, X'_1)$ is cartesian, the composition $g' \circ c(\lambda, X'_0)$ factors as $c(\lambda, X'_1) \circ g$ for some unique $g : \lambda^*(X_0) \to \lambda^*(X_1)$, such that $p(g) = 1_B$. We extend λ^* to act on morphisms like g' by taking $\lambda^*(g') = g$. It can be checked that we obtain a functor in this way.

Establishing that a functor is a fibration can be aided by the following observation, which is not hard to show: a functor $p : \mathbf{X} \to \mathbf{B}$ is a fibration with cleavage c if $c(\lambda, X')$ is cartesian for $\lambda : B \to B'$ in $\mathbf{B}$ and $X' \in p^{-1}(B')$ and moreover that the composition of cartesian morphisms given by c is cartesian (the composition need not however be precisely that cartesian lifting given by c).

Similarly, a cofibration $p : \mathbf{X} \to \mathbf{B}$ with an explicit choice of cocartesian liftings $d(X, \lambda) : X \to \lambda_!(X)$, for a morphism $\lambda : B \to B'$ in $\mathbf{B}$ and object $X \in p^{-1}(B)$, yields a functor $\lambda_! : p^{-1}(B) \to p^{-1}(B')$ extending the construction $\lambda_!(X)$ on objects X. In general, a function like d providing a choice of cocartesian liftings will be called a *cocleavage* of the associated cofibration. For any $\lambda : B \to B'$, the cofibration p and its cocleavage d determine a functor $\lambda_! : p^{-1}(B) \to p^{-1}(B')$ which acts on morphisms as follows:
There are cocartesian morphisms

$$d(X_0, \lambda) : X_0 \to \lambda_!(X_0) \text{ and } d(X_1, \lambda) : X_1 \to \lambda_!(X_1).$$

As $d(X_0, \lambda)$ is cocartesian, any morphism $g : X_0 \to X_1$ in $p^{-1}(B)$ determines a unique morphism $g' : \lambda_!(X_0) \to \lambda_!(X_1)$ in $p^{-1}(B')$ such that $d(X_1, \lambda) \circ g = g' \circ d(X_0, \lambda)$. This morphism g' we take as the value of $\lambda_!(g)$. In showing that a functor $p : \mathbf{X} \to \mathbf{B}$ is a cofibration with cocleavage d it is sufficient to show that $d(X, \lambda)$ is cocartesian for $\lambda : B \to_* B'$ in $\mathbf{B}$ and $X \in p^{-1}(B)$ and moreover that the composition of cocartesian morphisms given by d is cocartesian.

It follows from the next result, that the two functors $\lambda_!$ and λ^* between fibres, arising from a morphism λ in the base category of a bifibration, are adjoint to each other.

Lemma 79 *Suppose $p : \mathbf{X} \to \mathbf{B}$ is a bifibration. Let $\lambda : A \to B$ be a morphism in $\mathbf{B}$. Functors $\lambda_! : p^{-1}(A) \to p^{-1}(B)$ determined by a cocleavage for p and $\lambda^* : p^{-1}(B) \to p^{-1}(A)$ determined by a cleavage for p form an adjunction between fibres in which $\lambda_!$ is left adjoint to λ^*.*

Proof: Assume $p : \mathbf{X} \to \mathbf{B}$ forms a bifibration, *i.e.* it is simultaneously a fibration and cofibration. Let $\lambda : A \to B$ be a morphism in the base category $\mathbf{B}$.

Let $X \in p^{-1}(A)$ and $Y \in p^{-1}(B)$. Write $r : \lambda^*(Y) \to Y$ for the cartesian lifting of λ with respect to Y given by the cleavage for the fibration. Write $l : X \to \lambda_!(X)$ for the cocartesian lifting of λ with respect to X given by the cocleavage for the cofibration.

Given a vertical morphism $f : \lambda_!(X) \to Y$, the fact that r is cartesian entails the existence of a unique vertical morphism $\phi(f) : X \to \lambda^*(Y)$ such that

$$r \circ \phi(f) = f \circ l.$$

The additional fact that l is cocartesian ensures ϕ is a bijection from morphisms $\lambda_!(X) \to Y$ in $p^{-1}(B)$ to morphisms $X \to \lambda^*(Y)$ in $p^{-1}(A)$.

To show $\lambda_!$ is left adjoint to λ^* we require that the bijection ϕ satisfies the following naturality conditions (see [18] p.79):

$$\text{(i) } \phi(k \circ f) = \lambda^*(k) \circ \phi(f), \qquad \text{(ii) } \phi(f \circ \lambda_!(h)) = \phi(f) \circ h,$$

for all $f : \lambda_!(X) \to Y$, $k : Y \to Y'$ in $p^{-1}(B)$ and $h : X' \to X$ in $p^{-1}(A)$.

In showing (i), let $r' : \lambda^*(Y') \to Y'$ be the cartesian lifting of λ with respect to Y' given by the cleavage. From the definition of the functor λ^* we see

$$r' \circ \lambda^*(k) = k \circ r.$$

From the definition of ϕ, we see

$$r \circ \phi(f) = f \circ l.$$

Hence

$$r' \circ (\lambda^*(k) \circ \phi(f)) = k \circ r \circ \phi(f) = k \circ f \circ l.$$

Furthermore, $\lambda^*(k) \circ \phi(f)$ is vertical because both $\lambda^*(k)$ and $\phi(f)$ are. But $\phi(k \circ f)$ is defined to be the unique vertical morphism such that

$$r' \circ \phi(k \circ f) = k \circ f \circ l.$$

Hence $\phi(k \circ f) = \lambda^*(k) \circ \phi(f)$, so fulfilling (i).

The argument for (ii) is analogous. Let $l' : X' \to \lambda_!(X')$ denote the cocartesian lifting of λ given by the cocleavage. From the definition of $\lambda_!$ as a functor, we get

$$\lambda_!(h) \circ l' = l \circ h$$

From the definition of ϕ, we have

$$r \circ \phi(f) = f \circ l.$$

Combining these facts we obtain

$$r \circ (\phi(f) \circ h) = f \circ l \circ h = (f \circ \lambda_!(h)) \circ l'$$

Clearly $\phi(f) \circ h$ is vertical. The definition of ϕ characterising $\phi(f \circ \lambda_!(h))$ as the unique vertical morphism g such that $r \circ g = (f \circ \lambda_!(h)) \circ l'$ implies the naturality condition (ii).

We conclude that $\lambda_!$ is left adjoint to λ^*. ∎

Suppose $p : \mathbf{X} \to \mathbf{B}$ is a fibration. The following result shows how products in $\mathbf{X}$ are related to fibre products (*i.e.* products in a fibre). Say a category has I-products, for a set I, if it has all products of size I.

Lemma 80 *Assume $p : \mathbf{X} \to \mathbf{B}$ is a fibration. Assume the base category $\mathbf{B}$ and the fibres $p^{-1}(B)$, for B in $\mathbf{B}$, have I-products. Assume functors $\lambda^* : p^{-1}(B') \to p^{-1}(B)$, for $\lambda : B \to B'$ in $\mathbf{B}$, determined by a cleavage for p, preserve products. Then the category $\mathbf{X}$ has I-products given in the following way:*
Let $X_i \in p^{-1}(B_i)$ for $i \in I$. Let $B, \lambda_i : B \to B_i$ for $i \in I$, be a product in $\mathbf{B}$. Let $c_i : \lambda_i^*(X_i) \to X_i$ be the cartesian liftings of λ_i given by the cleavage. Let X, q_i where $i \in I$, be a product of the objects $\lambda_i^*(X_i)$, where $i \in I$, in the fibre $p^{-1}(B)$. Then X, $c_i \circ q_i$ for $i \in I$, is a product in $\mathbf{X}$.

Proof: Under the assumptions stated, we prove that X with projections $c_i \circ q_i$, for $i \in I$, is a product.

Suppose $X' \in p^{-1}(B')$ and $f_i : X' \to X_i$ are morphisms in $\mathbf{X}$ for all $i \in I$. Projecting to the base category we obtain a family of morphisms $p(f_i) : B' \to B_i$, for $i \in I$, in $\mathbf{B}$. As B, λ_i where $i \in I$, is a product in $\mathbf{B}$ there is a unique $\lambda : B' \to B$ such that

$$p(f_i) = \lambda_i \circ \lambda$$

for all $i \in I$.

Associated with λ are the cartesian liftings

$$d_i : \lambda^*\lambda_i^*(X_i) \to \lambda_i^*(X_i) \text{ for } i \in I$$

and

$$d : \lambda^*(X) \to X$$

specified by the cleavage of the fibration. For all $i \in I$, the definition of how λ^* acts on morphisms gives

$$d_i \circ \lambda^*(q_i) = q_i \circ d. \tag{1}$$

For each $i \in I$ the composition $c_i \circ d_i : \lambda^*\lambda_i(X_i) \to X_i$ of cartesian morphisms is itself cartesian. Hence for each i the morphism f_i factors as

$$c_i \circ d_i \circ f_i' = f_i \tag{2}$$

for a unique vertical morphism $f_i' : X' \to \lambda^*\lambda_i^*(X_i)$. But, by assumption λ^* preserves products so $\lambda^*(X)$ with projections $\lambda^*(q_i)$, for $i \in I$, is a product in $p^{-1}(B')$. Thus there is a unique morphism $f' : X' \to \lambda^*(X)$ in $p^{-1}(B')$ such that

$$\lambda^*(q_i) \circ f' = f_i' \tag{3}$$

for all $i \in I$. Hence we obtain a morphism $f = d \circ f' : X' \to X$ for which

$$\begin{aligned} c_i \circ q_i \circ f &= c_i \circ q_i \circ d \circ f' && \text{by definition} \\ &= c_i \circ d_i \circ \lambda^*(q_i) \circ f' && \text{by (1)} \\ &= c_i \circ d_i \circ f_i' && \text{by (3)} \\ &= f_i && \text{by (2)} \end{aligned}$$

for all $i \in I$.

Indeed morphisms $g : X' \to X$ are uniquely determined by the property that

$$c_i \circ q_i \circ g = f_i \qquad \text{for all } \in I \tag{4}$$

Assume g satisfies property (4). As d is cartesian, g factors as

$$g = d \circ g' \tag{5}$$

for a vertical $g' : X' \to \lambda^*(X)$. Now, for each $i \in I$,

$$\begin{aligned} c_i \circ d_i \circ (\lambda^*(q_i) \circ g') &= c_i \circ q_i \circ d \circ g' \qquad \text{by (1)} \\ &= c_i \circ q_i \circ g \qquad \text{by (5)} \\ &= f_i \qquad \text{by (4).} \end{aligned}$$

But recall (2) which says: for each i

$$c_i \circ d_i \circ f_i' = f_i \tag{2}$$

for a unique vertical morphism $f_i' : X' \to \lambda^*\lambda_i^*(X_i)$. Hence $\lambda^*(q_i) \circ g' = f_i'$ for all $i \in I$. The fact that $\lambda^*(X), \lambda^*(q_i)$ for $i \in I$, is a product in $p^{-1}(B')$ ensures $g' = f'$ and so that $g = f$.

We conclude that X, with projections $c_i \circ q_i$ for $i \in I$, is a product in $\mathbf{X}$. This case is typical and shows that $\mathbf{X}$ has products generally. ∎

There is of course a dual result for coproducts and a cofibration.

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 which 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 $L : \mathbf{X} \to \mathbf{Y}$ and $R : \mathbf{Y} \to \mathbf{X}$ form an adjunction with L left adjoint to R. The adjunction is said to be *fibrewise*, with respect to p and q, iff $q \circ L = p$ and $p \circ R = q$ and each component of the counit $\epsilon_Y : LR(Y) \to Y$ is vertical, for $Y \in \mathbf{Y}$, *i.e.* $q(\epsilon_Y) = 1_{q(Y)}$ (or equivalently, components of the unit are vertical).

Lemma 81 *Suppose $p : \mathbf{X} \to \mathbf{B}$ and $q : \mathbf{Y} \to \mathbf{B}$ and that functors $L : \mathbf{X} \to \mathbf{Y}$ and $R : \mathbf{Y} \to \mathbf{X}$ form a fibrewise adjunction with L left adjoint to R. Then the right adjoint R preserves strong cartesian morphisms, and cartesian morphisms.*

Proof: We only show strong cartesian morphisms are preserved—showing the preservation of cartesian morphisms is similar.

Suppose $f : Y \to Y'$ is strong cartesian and $q(f) = \lambda : B \to B'$. We need that $R(f) : R(Y) \to R(Y')$ is strong cartesian over $p(R(f)) = \lambda$. Let $g' : X \to R(Y')$ be such that $p(g') = \lambda' : B'' \to B'$ and $\lambda' = \lambda \circ \lambda''$ for $\lambda'' : B'' \to B$. We are required to show

$$\exists! g'' : X \to R(Y).\ p(g'') = \lambda'' \ \&\ g' = (R(f)) \circ g''. \tag{$*$}$$

Let

$$\epsilon : LR(Y) \to Y \text{ and } \epsilon' : LR(Y') \to Y'$$

be components of the counit of the adjunction. Consider the morphism $\epsilon' \circ (L(g')) : L(X) \to Y'$. We have

$$q(\epsilon' \circ (L(g'))) = q(\epsilon') \circ q(L(g')) = 1'_B \circ p(g') = \lambda'.$$

As $f : Y \to Y'$ is strong cartesian there is a unique $k : L(X) \to Y$ with $q(k) = \lambda''$ and

$$f \circ k = \epsilon' \circ (L(g')).$$

Now by the cofreeness of $\epsilon : LR(Y) \to Y$, there is a unique $g'' : X \to R(Y)$ such that

$$\epsilon \circ (L(g'')) = k.$$

Also

$$p(g'') = qL(g'') = 1_B \circ (qL(g'')) = q(\epsilon \circ (L(g''))) = q(k) = \lambda''.$$

From the naturality of the adjunction we have

$$\epsilon' \circ (LR(f)) = f \circ \epsilon.$$

Using this fact we obtain

$$\begin{aligned} \epsilon' \circ L((R(f)) \circ g'') &= \epsilon' \circ (LR(f)) \circ (L(g'')) \\ &= f \circ \epsilon \circ (L(g'')) \\ &= f \circ k \\ &= \epsilon' \circ (L(g')). \end{aligned}$$

But $\epsilon' : LR(Y)' \to Y'$ is cofree, so by the accompanying uniqueness property we get

$$g' = (R(f)) \circ g''.$$

Thus we have fulfilled the existence part of the requirement $(*)$.

To show uniqueness, assume also that

$$g_1 : X \to R(Y) \ \& \ p(g_1) = \lambda'' \ \& \ g' = (R(f)) \circ g_1.$$

Then

$$\begin{aligned} \epsilon' \circ (L(g')) &= \epsilon' \circ L(R(f) \circ g_1) \\ &= \epsilon' \circ (LR(f)) \circ (L(g_1)) \\ &= f \circ \epsilon \circ (L(g_1)) \text{ by naturality of the adjunction.} \end{aligned}$$

Recall

$$\epsilon' \circ (L(g')) = f \circ k = f \circ \epsilon \circ (L(g'')).$$

Thus

$$f \circ (\epsilon \circ (L(g_1))) = f \circ (\epsilon \circ (L(g''))).$$

But f is strong cartesian so $\epsilon \circ (L(g_1)) = \epsilon \circ (L(g''))$. Finally by the cofreeness of ϵ we obtain $g_1 = g''$, as required for the uniqueness part of $(*)$. ∎

Note that lemma 81 does not state that the left adjoint L preserves cartesian morphisms. Nor does it entail that the right adjoint R 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. Recall, for us a coreflection is understood to be an adjunction in which the unit is a natural isomorphism. We state the results only for coreflections, though we will occasionally refer to the dual results for reflections.

Lemma 82 *Suppose* $p : \mathbf{X} \to \mathbf{B}$ *and* $q : \mathbf{Y} \to \mathbf{B}$ *and that functors* $L : \mathbf{X} \to \mathbf{Y}$ *and* $R : \mathbf{Y} \to \mathbf{X}$ *form a fibrewise coreflection with* L *left adjoint to* R. *Let* $f : X \to X'$ *be a morphism in* $\mathbf{X}$. *If* $L(f)$ *is strong cocartesian (respectively cocartesian) with respect to* q *then* f *is strong cocartesian (respectively cocartesian) with respect to* p.

Proof: We show that L reflects cocartesian morphisms—the proof that it reflects strong cocartesian morphisms is essentially the same.

Let $f : X \to X'$ be a morphism in $\mathbf{X}$. Assume $L(f)$ is cocartesian with respect to q. In order to show f is cocartesian assume $f' : X \to X''$ is a morphism of $\mathbf{X}$ for which $p(f') = p(f)$. Because $L(f) : L(X) \to L(X')$ is cocartesian there is a unique $g : L(X') \to L(X'')$ in $\mathbf{Y}$ such that

$$g \circ L(f) = L(f').$$

We require the existence of a unique $h : X' \to X''$ in $\mathbf{X}$ satisfying

$$h \circ f = f'. \tag{$\dagger$}$$

Suppose $h : X' \to X''$ in $\mathbf{X}$ satisfies $h \circ f = f'$. Then

$$L(h) \circ L(f) = L(f')$$

where

$$L(h) = g,$$

by the cocartesian nature of $L(f)$. Because of the unit of the adjunction is a natural isomorphism, say with components

$$\eta' : X' \to RL(X'),$$
$$\eta'' : X'' \to RL(X''),$$

we see

$$\eta'' \circ h = RL(h) \circ \eta' = R(g) \circ \eta'$$

and so

$$h = \eta''^{-1} \circ R(g) \circ \eta'.$$

Thus supposing $h : X' \to X''$ in $\mathbf{X}$ satisfies $h \circ f = f'$ implies $h = \eta''^{-1} \circ R(g) \circ \eta'$. This ensures the uniqueness of h satisfying ($\dagger$). Moreover, taking h to be $\eta''^{-1} \circ R(g) \circ \eta'$ it follows from the unit being a natural isomorphism that ($\dagger$) holds.∎

The next result is useful for pulling the property of being a fibration or cofibration across an coreflection from $\mathbf{X}$ to $\mathbf{Y}$. It describes how to construct cartesian and cocartesian liftings in $\mathbf{X}$ from those in $\mathbf{Y}$. The lemma and its proof apply equally well to cartesian and cocartesian, or strong cartesian and cocartesian morphisms.

Lemma 83 *Suppose $p : \mathbf{X} \to \mathbf{B}$ and $q : \mathbf{Y} \to \mathbf{B}$ and that functors $L : \mathbf{X} \to \mathbf{Y}$ and $R : \mathbf{Y} \to \mathbf{X}$ form a fibrewise coreflection with L left adjoint to R. Let η and ϵ be the unit and counit respectively of the adjunction. Let the map $f \mapsto \hat{f}$ from morphisms $f : L(X) \to Y$ in $\mathbf{Y}$ to morphisms $\hat{f} : X \to R(Y)$ in $\mathbf{X}$ be the natural bijection of the adjunction.*
(1) If q is a fibration then so is p. Moreover, if $c : Y \to L(X)$ is a (strong) cartesian morphism with respect to q, then $d =_{def} \eta_X^{-1} \circ R(c) : R(Y) \to X$ is (strong) cartesian with respect to p; the (strong) cartesian morphism d is the unique morphism $R(Y) \to X$ such that $L(d) = c \circ \epsilon_Y$.
(2) If q is a cofibration such that for all (strong) cocartesian morphisms $L(X) \to Y$ with respect to q, for $X \in \mathbf{X}, Y \in \mathbf{Y}$, the component $\epsilon_Y : LR(Y) \to Y$ is an isomorphism then p is a cofibration. Moreover, if $c : L(X) \to Y$ is (strong) cocartesian with respect to q and $\epsilon_Y : LR(Y) \to Y$ is an isomorphism then $\hat{c} : X \to R(Y)$ is (strong) cocartesian with respect to p.

Proof: (1) If $c : Y \to L(X)$ is (strong) cartesian such that $q(c) = \lambda$, then by lemma 81, $R(c) : R(Y) \to RL(X)$ is (strong) cartesian with respect to $p : \mathbf{X} \to \mathbf{B}$. But $\eta_X : X \cong RL(X)$, the unit of the coreflection at X is a vertical isomorphism, so the composition $d =_{def} \eta_X^{-1} \circ R(c) : R(Y) \to X$ is a (strong) cartesian morphism such that $p(d) = \lambda$. The alternative characterisation of d follows from the coreflection. From the adjunction we have $\epsilon_{L(X)} \circ L(\eta_X) = 1_{L(X)}$. But the adjunction is a coreflection, making η_X an isomorphism, so $\epsilon_{L(X)} = L(\eta_X^{-1})$. Thus

$$L(d) = L(\eta_X^{-1}) \circ LR(c) = \epsilon_{L(X)} \circ LR(c).$$

However, as ϵ is a natural transformation,

$$\epsilon_{L(X)} \circ LR(c) = c \circ \epsilon_Y.$$

Because of the coreflection, L is faithful (and full) so $d : R(Y) \to X$ is unique such that $L(d) = c \circ \epsilon_Y$.

Assume that q is a fibration. Let $\lambda : B \to B'$ be a morphism in the base category $\mathbf{B}$. Let $X \in \mathbf{X}$ with $p(X) = B'$. As q is a fibration, there is a is strong cartesian $c : Y \to L(X)$ such that $q(c) = \lambda$. As above, there is a strong cartesian morphism $d : R(Y) \to X$ with $p(d) = \lambda$. Hence $p : \mathbf{X} \to \mathbf{B}$ is a fibration.

(2) Let $X \in \mathbf{X}$, $Y \in \mathbf{Y}$ have the property that

$$c : L(X) \to Y$$

is (strong) cocartesian with respect to q. The counit at Y,

$$\epsilon_Y : LR(Y) \cong Y$$

is assumed to be a vertical isomorphism. By cofreeness, there is a unique morphism

$$\hat{c} : X \to R(Y)$$

in $\mathbf{X}$ such that

$$c = \epsilon \circ L(\hat{c}).$$

But then $L(\hat{c}) = \epsilon_Y^{-1} \circ c$ a composition of a vertical isomorphism with a (strong) cocartesian morphism. Hence $L(\hat{c})$ is itself (strong) cocartesian. Now by lemma 82 it follows that $\hat{c}$ is (strong) cocartesian too.

It follows that if q is a cofibration such that for all (strong) cocartesian morphisms $L(X) \to Y$ with respect to q the component $\epsilon_Y : LR(Y) \to Y$ is an isomorphism then p is a cofibration.∎

Appendix B

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 $f : X \rightarrow_* Y$ for a function $f : X \cup \{*\} \rightarrow Y \cup \{*\}$ such that $f(*) = *$. When $f(x) = *$, for $x \in X$, we say $f(x)$ is *undefined* and otherwise *defined*. We say $f : X \rightarrow_* Y$ is *total* when $f(x)$ is defined for all $x \in X$. Of course, such total morphisms $X \rightarrow_* Y$ correspond to the usual total functions $X \rightarrow Y$, with which they shall be identified. For the category $\mathbf{Set}_*$, we take as objects sets which do not contain $*$, and as morphisms functions $f : X \rightarrow_* Y$, 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.

Appendix 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 2.3 (part (a)) and, by a slightly more general argument, section 3.8.3 (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 $\mid t \mid$ 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.

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

$$\mid\; t \;\mid = (tag(t), size(t))$$

where the functions tag and size are defined by the following structural inductions:

$tag(nil)$	$=$	$tag(x) = 0$	$size(nil)$	$=$	$size(x) = 0$
$tag(at)$	$=$	$tag(t)$	$size(at)$	$=$	$1 + size(t)$
$tag(t_0 \oplus t_1)$	$=$	$min(tag(t_0), tag(t_1))$	$size(t_0 \oplus t_1)$	$=$	$1 + size(t_0) + size(t_1)$
$tag(t_0 \times t_1)$	$=$	$tag(t_0) + tag(t_1)$	$size(t_0 \times t_1)$	$=$	$1 + size(t_0) + size(t_1)$
$tag(t \restriction \Lambda)$	$=$	$tag(t\{\Xi\}) = tag(t)$	$size(t \restriction \Lambda)$	$=$	$size(t\{\Xi\}) = 1 + size(t)$
$tag(rec\, x.t)$	$=$	$tag(t)$	$size(rec\, x.t)$	$=$	$1 + size(t)$
$tag((l,t))$	$=$	$1 + tag(t)$.	$size((n,t))$	$=$	$1 + size(t)$

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

$$\mid t[rec\, x.t/x] \mid < \mid t' \mid .$$

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

$$tag(rec\, x.t) = tag(t) < \mid + tag(t') = tag((2,t')).$$

Hence

$$| rec\ x.t | < | (2, t') | .$$

holds of the transition $rec\ x.t \xrightarrow{a} (2, t')$.∎

(a). A uniqueness property of guarded recursions in T

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.

Definition: Let $T = (S, i, L, tran)$ be a transition system. Define $T^{(k)}$ to be $(S', i, L', tran')$ 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 and L' is the subset of labels $a \in L$ for which there is a transition $(s, a, s') \in tran'$.

Let $f = (\sigma, \lambda) : T_0 \to T_1$ be a morphism of transition systems. 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 $T_0^{(k)}$.

Lemma 84 *Suppose $T_{n,m}$ are transition systems for $n, m \in \omega$ with the property that $T_{n,m} \trianglelefteq T_{n',m'}$ when $n \leq n'$ 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 85

1. *For each $k \in \omega, (-)^{(k)}$ is a functor on the category* **T** *of transition systems; it restricts to an endofunctor on the subcategory where all morphisms are monics.*

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 2.3, 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 monomorphisms.

(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

$$\begin{array}{rcll} \mathcal{R}(\bigcup_n T_n) & = & \bigcup_k (\bigcup_n T_n)^{(k)} & \text{by the definition of } \mathcal{R} \\ & = & \bigcup_k \bigcup_n T_n^{(k)} & \text{by continuity of } (-)^{(k)} \\ & = & \bigcup_n \bigcup_k T_n^{(k)} & \text{by lemma 84} \\ & = & \bigcup_n \mathcal{R}(T_n) & \text{by the definition of } \mathcal{R}. \blacksquare \end{array}$$

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 monomorphisms on transition systems. These facts extend to morphisms. A functor F defined by a guarded recursion has the property that

$$(F(f))^{(k)} = (F(f^{(k-1)}))^{(k)} \tag{3}$$

on monomorphisms of transition systems f.

Definition: We will call a functor F on the category of transition systems with monomorphisms *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:

$$T \cong \mathcal{R} \circ F(T) \Rightarrow T \cong \mathcal{R}(fix(F)).$$

Proof: Any functor 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

$$\begin{array}{lcll} (F^k(I))^{(n+1)} & = & (F(F^{k-1}(I)))^{(n+1)} & \\ & = & (F((F^{k-1}(I))^{(n)}))^{(n+1)} & \text{as } F \text{ is guarded} \\ & = & (F((fix(F))^{(n)})^{(n+1)} & \text{from the induction hypothesis as } k-1 \geq n \\ & = & (F(fix(F)))^{(n+1)} & \text{as } F \text{ is guarded} \\ & = & (fix(F))^{(n+1)}. & \end{array}$$

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, reason, for an arbitrary transition system T, that

$$\begin{array}{lcll} \mathcal{R} \circ F \circ \mathcal{R}(T) & = & \mathcal{R} \circ F(\bigcup_k T^{(k)}) & \text{by definition of } \mathcal{R}, \\ & = & \bigcup_k \mathcal{R} \circ F(T^{(k)}) & \text{by continuity of } \mathcal{R} \text{ and } F, \\ & = & \bigcup_k \bigcup_n (F(T^{(k)}))^{(n)} & \text{by definition of } \mathcal{R}, \\ & = & \bigcup_k (F(T^{(k)}))^{(k)} & \text{by lemma 84} \\ & = & \bigcup_k (F(T))^{(k)} & \text{by (1)} \\ & = & \mathcal{R} \circ F(T) & \text{by definition of } \mathcal{R} \end{array}$$

It follows that

$$\begin{array}{lcll} \mathcal{R}(fix(F)) & = & \mathcal{R}(\bigcup_n F^n(I)) & \\ & = & \bigcup_n \mathcal{R}\, F^n(I) & \\ & = & \bigcup_n (\mathcal{R} \circ F)^n \mathcal{R}(I) & \text{by repeated use of (5)} \\ & = & \bigcup_n (\mathcal{R} \circ F)^n(I) & \\ & = & fix(\mathcal{R} \circ F). & \end{array}$$

Hence, writing $G = \mathcal{R} \circ F$, we can restate the goal of our proof (in the statement of the theorem) as

$$T \cong G(T) \Rightarrow T \cong fix(G) \tag{†}$$

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

$$u : I \to T.$$

By induction on u 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

$$\begin{aligned}(G^{n+1}(u))^{(n+1)} &= (G(G^n(u)))^{(n+1)} \\ &= (G((G^n(u))^{(n)}))^{(n+1)},\end{aligned}$$

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

$$\begin{aligned}(G^n(u))^{(n)} &= (G^n(\theta))^{(n)} \circ (G^{n+1}(u))^{(n)} \circ (G^n(j))^{(n)} \\ &= (G^n(\theta))^{(n)} \circ (G^{n+1}(u))^{(n)}\end{aligned}$$

It follows that

$$\begin{aligned}\phi_n &= (\theta_n \circ G^n(u))^{(n)} \\ &= (\theta \circ \cdots \circ G^{(n-1)}(\theta))^{(n)} \circ (G^n(\theta))^{(n)} \circ (G^{n+1}(u))^{(n)} \\ &= ((\theta_{n+1} \circ G^{n+1}(u))^{(n+1)})^{(n)} \text{ by proposition 85} \\ &= \phi_{n+1}^{(n)}.\end{aligned}$$

It follows that

$$\phi = \bigcup_{n \in \omega} \phi_n$$

is an isomorphism $fix(G) \to T$.∎

(b). A uniqueness property of guarded recursions in TI

This part of the Appendix is dedicated to showing the following uniqueness property of the semantics of the process terms as transition systems with independence:
Lemma 75 If x is guarded in a process term t, then

$$T \cong \mathcal{R}(\mathbf{TI}[\![t]\!]\rho[T/x]) \Rightarrow T \cong \mathcal{R}(\mathbf{TI}[\![rec\, x.t]\!]\rho),$$

for any environment ρ assigning transition systems with independence to process variables.

The proof closely follows the lines of Appendix C (a), but where the extra axioms required by transition systems with independence complicate the basic definitions. Again, a key idea is that of a functor $(-)^{(k)}$ "projecting" a transition system this time with independence, to that part of it which is reachable within k steps. The simple definition used in part (a) is not satisfactory here however because, as it stands, it generally does not yield a transition system with independence $T^{(k)}$ such that $T^{(k)} \trianglelefteq T$, from one T. To obtain a suitable generalisation we take $T^{(k)}$ to be the $\trianglelefteq$-least transition system with independence which includes that part of T reachable within k steps and for which $T^{(k)} \trianglelefteq T$.

Proposition 86 *(i) Let T be a transition system with independence. There is a $\trianglelefteq$-least transition system with independence $T^{(k)}$ such that $T^{(k)} \trianglelefteq T$ and $T^{(k)}$ includes all those transitions and states reachable within k steps.*

(ii) Suppose $(\sigma, \lambda) : T \to T'$ is a morphism in **TI** *then $(\sigma, \lambda)^{(k)} =_{def} (\sigma', \lambda') : T^{(k)} \to T'^{(k)}$ is a morphism in* **TI** *where σ' is the restriction of σ to the states of $T^{(k)}$ and λ' the restriction of λ to the labels of $T^{(k)}$. This makes $(-)^{(k)}$ a functor on* **TI** *which moreover restricts to a functor on the subcategory of monics.*

Proof: (i) Let T be a transition system with independence $(S, i, L, Tran, I)$. For $k \in \omega$, take T_0 to be $(S_0, i, L_0, Tran_0, I_0)$ consisting of states S_0 and transitions $Tran_0$ of T reachable within k steps with L_0 all labels appearing in $Tran_0$ and $I_0 = I \cap Tran_0^2$ (In general T_0 will not be a transition system with independence.) Refer to the axioms on transition systems with independence. Because of axiom (1), axioms (2) and (3) entail T satisfies:

(2)' $(s, a, s')I(s, b, s_2)$ implies there are *unique* transitions $(s_1, b, u), (s_2, a, u) \in Tran$ such that $(s, a, s_1)I(s_1, b, u)$ & $(s, b, s_2)I(s_2, a, u)$.

(3)' $(s,a,s_1)I(s_1,b,u)$ implies there are *unique* transitions $(s,b,s_2),(s_2,a,u) \in Tran$ such that $(s,a,s_1)I(s,b,s_2)$ & $(s,b,s_2)I(s_2,a,u)$.

Inductively close T_0 up under those unique states and transitions required of the independence relation by (2)' and (3)'—at each stage take the independence relation to be the restriction of that of T. This inductive construction yields $T^{(k)}$ the $\trianglelefteq$-least transition system with independence—axioms (2) and (3) are ensured by the construction while (1) and (4) clearly hold—which includes T_0 and gives $T^{(k)} \trianglelefteq T$.
(ii) It is clear that if $f : T \to T'$ is a morphism in **TI** then so is $f^{(k)} : T^{(k)} \to T'$, for $k \in \omega$. We further require that the restriction, $f^{(k)}$, of f to $T^{(k)}$ has its image in $T'^{(k)}$. Consider the inductive definition of $T^{(k)}$. Because morphisms in **TI** preserve or collapse transitions to idle transitions, and furthermore preserve the independence of transitions any stage in inductive construction of $T^{(k)}$ has its image under f within the corresponding stage in the inductive construction of $T'^{(k)}$. The remaining properties are clear. ∎

Proposition 86 provides the appropriate definition of the family of functors $(-)^{(k)}$: **TI** $\to$ **TI** 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 transition systems with independence is definable if it acts on T so that

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

for some choice of process term t and variable x. Any such definable operation is $\trianglelefteq$-continuous. It will also satisfy

$$(F(T))^{(k)} = (F(T^{(k)}))^{(k)}, \text{for } k \in \omega,$$

—just as in part (a). However, now the proof of this fact is more intricate. In particular, it requires that we show

$$(T \times U)^{(k)} = (T^{(k)} \times U^{(k)})^{(k)},$$

for transition systems with independence T and U. As $T^{(k)} \trianglelefteq T$ and $U^{(k)} \trianglelefteq U$, by monotonicity we deduce

$$(T^{(k)} \times U^{(k)})^{(k)} \trianglelefteq (T \times U)^{(k)}.$$

For the converse inclusion, we remark that

$$T^{(k)} \times U^{(k)} \trianglelefteq T \times U$$

and, moreover, that $T^{(k)} \times U^{(k)}$ includes that part of $T \times U$ reachable within k steps. But by proposition 86(i), this entails

$$(T \times U)^{(k)} \trianglelefteq T^{(k)} \times U^{(k)}$$

and hence that

$$(T \times U)^{(k)} \trianglelefteq (T^{(k)} \times U^{(k)})^{(k)}.$$

As in part (a),

$$(a(T))^{(k)} = (a(T^{(k-1)}))^{(k)}$$

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 a functor F definable by a process term satisfies

$$(F(f))^{(k)} = (F(f^{(k)}))^{(k)},$$

on monics 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 "transition system with independence" instead of "transition system"; the proofs of proposition 85 and lemma 9 are sufficiently abstract to apply equally well in the more general situation of transition systems with independence. This furnishes a proof of the uniqueness property of guarded recursions in **TI**.

Bibliography

[1] Advanced course on Petri nets. LNCS 254,255, Springer, 1987.

[2] Bénabou,J., Fibred categories and the foundations of naive category theory. JSL 50, 1985.

[3] Barr, M., and Wells, C., Category theory for Computer Science. Prentice Hall, 1990.

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

[5] Berry, G, Modèles completement adéquats et stables des λ-calculs typées. Thèse de Doctorat d'Etat, Université Paris VII, 1979.

[6] Brookes, S.D., On the axiomatic treatment of concurrency. LNCS 197,1985.

[7] Brookes, S.D., On the relationship of CCS and CSP. Proc. of ICALP 1983, LNCS 154, Springer, 1983.

[?] Boudol, G., and Castellani, I., Flow models of distributed computations: three equivalent semantics for CCS. INRIA Research Report 1484, 1991.

[8] Campbell,R.H, Habermann, A.N, The specification of process synchroniation by path expressions.LNCS 16,1973.

[9] Johnstone,P, Fibred categories.Lecture notes,Cambridge Univ,1983.

[10] Hoare,C.A.R., A model for communicating sequential processes. Technical Report PRG-22, Programming Research Group, University of Oxford Computing Lab., 1981.

[11] Hoare,C.A.R, Brookes,S.D, and Roscoe,A.W, A Theory of Communicating Processes. JACM,1984.

[12] Hoare,C.A.R, Olderog,E, Specification oriented semantics for communicating processes.LNCS 154,1983.

[13] Hennessy, M., Algebraic theory of processes. MIT Press, 1988.

[14] Kahn, G., and Plotkin, G., Structures de données concrètes. IRIA-Laboria Report 336, 1979.

[15] Keller, R.M., Formal verification of parallel programs. Communications of the ACM, no.19, vol.7, pp.371-384, 1976.

[16] Labella,A, Petterossi,A, Categorical models of process cooperation.LNCS 240,1985.

[17] Lauer,P, Campbell,R.H, Formal semantics for a class of high level primitives for coordinating concurrent processes. Acta Inf.5,1974.

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

[19] Mukund, M., and Nielsen, M., CCS, locations and asynchronous transition systems. DAIMI Report, to appear in the proceedings of Fonndations of Software Technology and Theoretical Computer Science, Springer Lecture Notes in C.S., 1992.

[20] Mazurkiewicz, A., Basic notions of trace theory. In the lecture notes for the REX summerschool in temporal logic, May 88, in Springer Lecture Notes in C.S., vol.354.

[21] Milner,A.R.G,Calculus of communicating systems.LNCS 92, 1980.

[22] Milner,A.R.G, Communication and concurrency. Prentice Hall, 1989.

[23] Martin-Löf,P, The domain interpretation of type theory. Lecture notes, Göteborg, 1983.

[24] Nielsen, M., Plotkin, G., Winskel, G., Petri nets, Event structures and Domains, part 1. Theoretical Computer Science, vol. 13 (1981) pp. 85–108.

[25] Plotkin, G.D., "Structural operational semantics." Lecture Notes, DAIMI FN-19, Aarhus University, Denmark, 1981 (reprinted 1991).

[26] Poigné,A, Category theory and logic.LNCS 240,1985.

[27] Pratt, V.R., Modelling concurrency with partial orders. International Journal of Parallel Programming, 15,1, p.33-71, Feb. 1986.

[28] Shields, M.W., Concurrent machines. Computer Journal, vol.28, pp.449-465, 1985.

[29] Taylor,P., Recursive domains, indexed category theory and polymorphism. PhD thesis,Cambridge Univ,1987.

[30] Sassone, V., PhD thesis, University of Pisa.

[31] Winskel, G., Events in Computation. PhD thesis, University of Edinburgh, available as a Comp. Sc. report (1980).

[32] Winskel,G, Event structure semantics of CCS and related languages. ICALP '82, LNCS 140,1982. A full version with proofs appears as a DAIMI, Report University of Aarhus, 1983.

[33] Winskel, G., A representation of completely distributive algebraic lattices. Report of the Computer Science Dept., Carnegie-Mellon University (1983).

[34] Winskel,G, Synchronisation trees. TCS, May 1985.

[35] Winskel,G, Categories of Models for Concurrency. LNCS 197,1984.

[36] Winskel,G, Petri nets, algebras, morphisms and compositionality. Information and Computation, March 1987.

[37] Winskel,G, Event structures. LNCS 255, 1987.

[38] Winskel, G., An introduction to event structures. In the lecture notes for the REX summerschool in temporal logic, May 88, in Springer Lecture Notes in C.S., vol.354.

[39] Winskel, G., A category of labelled Petri nets and compositional proof system. In the proceedings of the the conference "Logic in Computer Science", Edinburgh, July 1988.