

Number 58



**UNIVERSITY OF  
CAMBRIDGE**

Computer Laboratory

## Categories of models for concurrency

Glynn Winskel

October 1984

15 JJ Thomson Avenue  
Cambridge CB3 0FD  
United Kingdom  
phone +44 1223 763500  
<http://www.cl.cam.ac.uk/>

© 1984 Glynn Winskel

Technical reports published by the University of Cambridge  
Computer Laboratory are freely available via the Internet:

*<http://www.cl.cam.ac.uk/techreports/>*

ISSN 1476-2986

# CATEGORIES OF MODELS FOR CONCURRENCY

by

*Glynn Winskel*

University of Cambridge,  
Computer Laboratory,  
Corn Exchange Street,  
Cambridge CB2 3QG.

## **Abstract.**

It is shown how a variety of models for concurrent processes can be viewed as categories in which familiar constructions turn out to be significant categorically. Constructions to represent various parallel compositions are often based on a product construction for instance. In many cases different models can be related by a pair of functors forming an adjunction between the two categories. Because of the way in which such pairs of functors preserve categorical constructions, the adjunction serves to translate between the different models, so it is seen how semantics expressed in terms of one model translates to semantics in terms of another.

## Introduction.

The theory of sequential programming language is well understood, making it possible to reason, often in a formal proof system, about the behaviour of programs. However the situation is less settled in the case of parallel programs, where several processes run concurrently to cooperate on a common task. There are concrete models like *Petri nets*, *event structures*, *synchronisation trees* and *state-transition systems*, which essentially model a process as moving through states as events occur. These models do not even represent concurrency in the same way. Models like Petri nets and event structures represent concurrent activity in terms of causal independence while most other models simulate concurrency by nondeterministic interleaving of atomic actions. Then there are more abstract models, perhaps based on *powerdomains* or on some reasonable idea of *operational equivalence*, formed with the idea of detecting and proving rather specific properties of processes. Other approaches are based on variants of modal logic but here too similar choices are faced; what is the underlying concrete model and how expressive should the modal logic be?

I am interested in developing a uniform mathematical framework to relate the many different models that exist for parallel computation. At present I have had most success with concrete models for languages in the style of R. Milner's "*Calculus of Communicating Systems*" (CCS) and C.A.R. Hoare's "*Communicating Sequential Process*" (CSP) [W1,2,3].

To give the idea, each kind of concrete model (*e.g.* Petri nets are such a model) carries a notion of morphism appropriate to languages like CCS and CSP, which make it into a category. Then useful constructions within the model, like parallel composition, arise as categorical constructions accompanied by abstract characterisations. Relations between two different kinds of models, say nets and trees, are expressed as an adjunction, between say the category of nets and the category of trees. Because of the way functors of an adjunction preserve categorical constructions this gives a smooth translation between semantics in one model and semantics in another. This technique works for a wide range of parallel programming languages with a wide variety of communication disciplines—they can be expressed in a very general way using the idea of *synchronisation algebra* [W1,2].

The way in which constructs in the programming languages are modelled by categorical constructions leads to accompanying proof rules. But unfortunately because the models are so concrete, the proof rules do not immediately capture aspects of behaviour at the level of abstraction one wants. Still, even the more abstract models and their proof rules have at their basis a concrete model of one sort or another. It is helpful to have a clear understanding of the relationship there is amongst the diversity of such models. Given a modicum of category theory it can be expressed in a surprisingly clean way, as I hope will come across. The basic category theory used here can be found in [AM] or [Mac].

## 1. Languages for communicating processes—an abstract view.

A host of programming languages CCS, CSP, SCCS, CIRCAL, OCCAM, MEIJE, ESTEREL... are based on the idea that processes communicate by events of synchronisation.

Individually a process  $P_0$  is thought of as capable of performing certain events. Some of them may be communications with the environment and others may be internal actions. Set in parallel with another process  $P_1$  an event  $e_0$  of  $P_0$  might synchronise with an event  $e_1$  of  $P_1$ . Whether they do or not will of course depend on what kinds of events  $e_0$  and  $e_1$  are because  $P_0$  and  $P_1$  can only perform certain kinds of synchronisation with their environments. But if they do synchronise we can think of them as forming a synchronisation event  $(e_0, e_1)$ . The synchronisation event  $(e_0, e_1)$  has the same effect on the process  $P_0$  as the component event  $e_0$  and similarly on  $P_1$  has the same effect as the event  $e_1$ .

Of course generally not all events of  $P_0$  will synchronise with events of  $P_1$ ; there might be an internal event of  $P_0$  for example which by its very nature cannot synchronise with any event of  $P_1$ . So we cannot expect all events of the parallel composition to have the form  $(e_0, e_1)$ . Some will have no component event from one process or the other. We can represent these events in the form  $(e_0, *)$  if the event  $e_0$  of  $P_0$  occurs unsynchronised with any event of  $P_1$  or  $(*, e_1)$  if the event  $e_1$  of  $P_1$  occurs unsynchronised. The  $*$  stands for the absence of an event from the corresponding component.

Thus we can view synchronisation as forming compound events from component events; a synchronisation event is viewed as a combination of events from the processes set in parallel.

Whether or not synchronisations can occur is determined by the nature of the events. Right from the start programming languages like CSP and CCS introduce notation to distinguish the different kinds of events. In the early version of CSP [H] the sequential processes are named and events of a sequential process are of the form: send a value  $v$  to process  $P$ , written  $P!v$ , or receive a value from process  $P$  into variable  $x$ , written  $P?x$ . There is no notation in this language of CSP for the events of synchronisation between processes. In the new version of CSP [HBR] events are distinguished according to the port at which they occur, so processes set in parallel must agree on the occurrence of events at common ports (this goes to back to an early idea of path expressions [CH] and is also at the basis of G. Milne's calculus CIRCAL [Mi]). The idea of ports, places at which processes communicate with the environment, underlies the language of CCS too—see [M1]; originally, apart from  $\tau$  the labels of CCS  $\alpha, \beta, \dots$  and their complementary labels  $\bar{\alpha}, \bar{\beta}, \dots$  were thought of as port names so in forming the parallel composition of two processes only complementary ports, as indicated by the labels, were connected. So correspondingly in CCS only events carrying complementary labels can synchronise to from an event labelled by  $\tau$ . In [M1] value-passing was handled using events labelled  $\alpha v$  to represent the input of value  $v$  at port  $\alpha$  and  $\bar{\alpha}v$  to represent the output of value  $v$  at port  $\alpha$ . Their introduction foreshadowed the liberating step Milner took in the language

of synchronous CCS, generally called SCCS. There the labels (called actions) are regarded very abstractly and can be composed according to monoids of actions.

### Synchronisation algebras

We take an abstract line inspired by Milner's monoids of actions in SCCS. However synchronisation algebras are more general than monoids of actions because they can express which actions can and cannot occur asynchronously.

We label events of processes to specify how they interact with the environment, so associated with any particular synchronisation algebra is a particular parallel composition. By specialising to particular synchronisation algebras we can obtain a wide range of parallel compositions.

A *synchronisation algebra*,  $(L, \circ, *, 0)$ , consists of a binary, commutative, associative operation  $\circ$  on a set of labels which always includes two distinguished elements  $*$  and  $0$ . The binary operation  $\circ$  says how labelled events combine to form synchronisation events and what labels such combinations carry. No real events are ever labelled by  $*$  or  $0$ . However their introduction allows us to specify the way labelled events synchronise without recourse to partial operations on labels. It is required that  $L \setminus \{*, 0\} \neq \emptyset$ .

The constant  $0$  is used to specify when synchronisations are disallowed. If two events labelled  $\lambda$  and  $\lambda'$  are not supposed to synchronise then their composition  $\lambda \circ \lambda'$  is  $0$ . For this reason  $0$  does indeed behave like a zero with respect to the "multiplication"  $\circ$  i.e.

$$\forall \lambda \in L. \lambda \circ 0 = 0.$$

In a synchronisation algebra, the constant  $*$  is used to specify when a labelled event can or cannot occur asynchronously. An event labelled  $\lambda$  can occur asynchronously iff  $\lambda \circ *$  is not  $0$ . We insist that the only divisor of  $*$  is  $*$  itself, essentially because we do not want a synchronisation event to disappear. We require

$$* \circ * = * \quad \text{and} \quad \forall \lambda, \lambda' \in L. \lambda \circ \lambda' = * \Rightarrow \lambda = *.$$

We present two synchronisation algebras as examples—more can be found in [W1,2].

**Example.** *The synchronisation algebra for CCS—no value passing:* In CCS [M1] events are labelled by  $\alpha, \beta, \dots$  or by their complementary labels  $\bar{\alpha}, \bar{\beta}, \dots$  or by the label  $\tau$ . The idea is that only two events bearing complementary labels may synchronise to form a synchronisation event labelled by  $\tau$ . Events labelled by  $\tau$  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.

$\circ$	*	$\alpha$	$\bar{\alpha}$	$\beta$	$\bar{\beta}$	$\dots$	$\tau$	0
*	*	$\alpha$	$\bar{\alpha}$	$\beta$	$\bar{\beta}$	$\dots$	$\tau$	0
$\alpha$	$\alpha$	0	$\tau$	0	0	$\dots$	0	0
$\bar{\alpha}$	$\bar{\alpha}$	$\tau$	0	0	0	$\dots$	0	0
$\beta$	$\beta$	0	0	$\tau$	0	$\dots$	0	0
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\dots$	$\cdot$	$\cdot$

**Example.** *The synchronisation algebra for  $\parallel$  in CSP:* In the new form of CSP—see [HBR, B]—events are labelled by  $\alpha, \beta, \dots$  or  $\tau$ . For its parallel composition  $\parallel$  events must “synchronise on”  $\alpha, \beta, \dots$ . In other words non- $\tau$ -labelled events cannot occur asynchronously. Rather, an  $\alpha$ -labelled event in one component of a parallel composition must synchronise with an  $\alpha$ -labelled event from the other component in order to occur; the two events must synchronise to form a synchronisation event again labelled by  $\alpha$ . The synchronisation algebra for this parallel composition takes the following form.

$\circ$	*	$\alpha$	$\beta$	$\dots$	$\tau$	0
*	*	0	0	$\dots$	$\tau$	0
$\alpha$	0	$\alpha$	0	$\dots$	0	0
$\beta$	0	0	$\beta$	$\dots$	0	0
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\dots$	$\cdot$	$\cdot$

Using synchronisation algebras one can define a generic programming language, inspired by CCS, SCCS and CSP but parameterised by the synchronisation algebra. For a synchronisation algebra  $L$ , the language  $\text{Proc}_L$  is given by the following grammar:

$$t ::= nil \mid x \mid \lambda t \mid t + t \mid t \otimes t \mid t[\Lambda \mid t[\Xi] \mid rec \ x.t$$

where  $x$  is in some set of variables  $X$  over processes,  $\lambda \in L \setminus \{*, 0\}$ ,  $\Lambda \subseteq L \setminus \{*, 0\}$ , and  $\Xi : L \rightarrow L$  is a relabelling function preserving  $*$  and  $0$  and such that  $\Xi(\lambda) = * \Rightarrow \lambda = *$  and  $\Xi(\lambda) = 0 \Rightarrow \lambda = 0$ —otherwise it would not lead to a sensible labelling of events.

We explain informally the behaviour of the constructs in the language  $\text{Proc}_L$ . The behaviour can be described accurately by the models presented in the next sections. Roughly, a process of  $\text{Proc}_L$  determines a pattern of event occurrences over time. The nature of the events, how they interact with the environment, is specified by associating each event with a label from the synchronisation algebra  $L$ . The term  $nil$  represents the  $nil$  process which has stopped and refuses to perform any event. A *guarded* process  $\lambda p$  first performs an event of kind  $\lambda$  to become the process  $p$ . A *sum*  $p + q$  behaves like  $p$  or  $q$ ; which branch of a sum is followed will often be determined by the context and what kinds of events the process is restricted to. A *parallel composition* process  $p \otimes q$  behaves like  $p$  and  $q$  set in

parallel. Their events of synchronisation are those pairs of events  $(e_0, e_1)$ , one from each process, where  $e_0$  is of kind  $\lambda_0$  and  $e_1$  is of kind  $\lambda_1$  so that  $\lambda_0 \circ \lambda_1 \neq 0$ ; the synchronisation event is then of kind  $\lambda_0 \circ \lambda_1$ . The restriction  $p[\Lambda]$  behaves like the process  $p$  but with its events restricted to lie in the set  $\Lambda$ . A relabelled process  $p[\Xi]$  behaves like  $p$  but with the events relabelled according to  $\Xi$ . A closed term  $\text{rec } x.p$  recursively defines a process  $x$  with body  $p$ .

The language  $\text{Proc}_L$  is not suited to value-passing. However it can easily be extended to be so, and again the discipline of communication can be handled using synchronisation algebras. Labels are taken to have two components  $\lambda v$ ; one,  $\lambda$ , can be thought of as a channel name and the other,  $v$ , as standing for the value passed or the value received. Processes can be parameterised in the manner of CCS with value-passing (see [M1]).



## 2. Petri nets, a general model.

Petri nets model processes in terms of how the occurrence of events incur changes in local states called conditions. This is expressed by a causal dependency (or flow) relation between sets of events and conditions, and it is this structure which determines the dynamic behaviour of nets once the causal dependency relation is given its natural interpretation. The most general Petri net we consider has the following form. Here and throughout this paper we refer the reader to the appendix for a treatment of multisets.

**Definition.** A Petri net is a 4-tuple  $(B, E, F, M_0)$  where

- (i)  $B$  is a non-null set of conditions,
- (ii)  $E$  is a set of events,
- (iii)  $F$  is a multiset of  $(B \times E) \cup (E \times B)$ , called the causal dependency relation
- (iv)  $M_0$  is a non-null multiset of conditions, called the initial marking

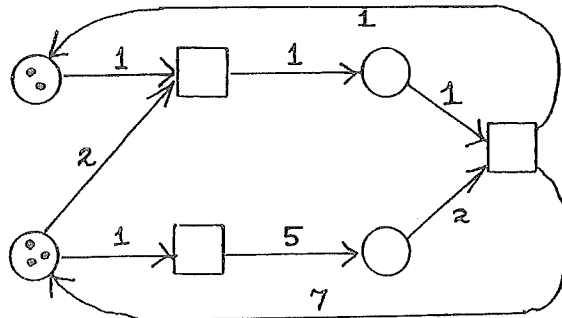
which satisfy the restrictions:

$$\forall e \in E \exists b \in B. F_{b,e} > 0 \quad \text{and} \quad \forall e \in E \exists b \in B. F_{e,b} > 0.$$

Thus we insist that each event causally depends on at least one condition and has at least one condition which is causally dependent on it.

Nets are often drawn as graphs in which events are represented as boxes and conditions as circles with directed arcs between them, weighted by positive integers, to represent the flow relation. The initial marking is represented by placing "tokens" of the appropriate multiplicity on each weighted condition. Here is an example.

**Example.**



By convention we understand an arc which carries no label to stand for an arc with weight 1.

Later, because we have some problems with both the interpretation and mathematics of such general nets we will restrict to a subclass. But let's be as general as possible for the moment.

**Nets viewed as algebras:** It is useful, both notationally and conceptually, to regard a Petri net as a 2-sorted algebra on multisets. This view underlies the techniques for finding invariants of nets by linear algebra [Pe, Br, R].

Nets are in 1-1 correspondence with 2-sorted algebras with sorts  $\mu B$  a non-trivial multiset of conditions and  $\mu E$  a multiset of events and operations the constant  $M_0 \in \mu B$ , and the unary operations  ${}^\circ(-) : E \rightarrow \mu B$  and  $(-)^{\circ} : E \rightarrow \mu B$  which satisfy

$$M_0 \neq \underline{0} \ \& \ (({}^\circ A = \underline{0} \ \text{or} \ A^{\circ} = \underline{0}) \Rightarrow A = \underline{0}).$$

### The dynamic behaviour of nets

States of a net are represented as *markings* which are simply multisets of conditions. You can think of a condition as a resource and its multiplicity as the amount of the resource. As an event occurs it consumes certain resources and produces others. What and how much is specified by the relation  $F$ . Continuing this interpretation, if there are enough resources then more than one event can occur concurrently, and it's even allowed that an event can occur to a certain multiplicity. Now this may be a little hard to swallow, and I'll come clean and admit that very soon we shall specialise to a subclass of nets, called *contact-free*, or *safe*, in which this cannot occur. The reason for specialising to safe nets has mathematical grounds—certain theorems don't go through otherwise—but this may well reflect something unnatural and obscure in the behaviour of the more general nets. I shall say more later.

Let  $N = (B, E, F, M_0)$  be a Petri net.

A marking  $M$  is a multiset of conditions, *i.e.*  $M \in \mu B$ .

Let  $M, M'$  be markings. Let  $A$  be a finite multiset of events. Define

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

This gives the *transition relation* between markings. When we wish to stress the net  $N$  in which the transition  $M \xrightarrow{A} M'$  occurs we write

$$N : M \xrightarrow{A} M'.$$

A *reachable marking* of  $N$  is a marking  $M$  for which  $M_0 \xrightarrow{A_0} M_1 \xrightarrow{A_1} \dots \xrightarrow{A_{n-1}} M_n = M$  for some markings and multisets of events.

Now we can define the special subclass of nets.

**Definition.** Say a Petri net  $N = (B, E, F, M_0)$  is *contact-free* iff  $F \leq \underline{1}$  and  $M \leq \underline{1}$  for all reachable markings  $M$ .

For contact-free nets we can write  $xFy$  instead of  $F_{x,y} = 1$ .

**Remark.** Often such nets are called *safe*. For them a condition only holds or fails to hold and an event either occurs or does not occur; they do not happen with multiplicities. For

these nets the term “condition” is consistent with its more usual use where it is imagined to assert a state of affairs which either holds or does not hold. In fact, often people go to the extent of using different terms, like “place” and “transition”, for the conditions and events of the general nets. In this short exposition there’s no need to be so finicky.

The behaviour of contact-free nets is particularly simple and can be expressed just with sets, without the use of multisets. Recall we identify sets with those multisets in which the multiplicity is 1 at greatest.

**Proposition.** *The behaviour of contact-free nets:*

*Let  $N = (B, E, F, M_0)$  be a contact-free net.*

*Let  $A$  be a set of events. Then  ${}^\circ A$  and  $A^\circ$  are sets too.*

*Any reachable marking is a set.*

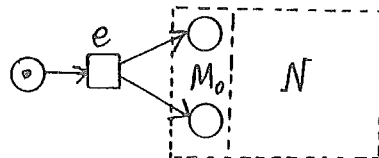
*Let  $M$  be a reachable marking. Let  $M'$  be a marking of  $N$ . Then  $M \xrightarrow{A} M'$  iff*

$$(\forall e \in A. {}^\circ e \subseteq M) \ \& \ (\forall e, e' \in A. {}^\circ e \cap {}^\circ e' = \emptyset) \ \& \ M' = (M \setminus {}^\circ A) \cup A^\circ.$$

For a contact-free net  $N$ , an event  $e$  is said to have *concession* at a reachable marking  $M$  if  ${}^\circ e \subseteq M$ . If two events  $e$  and  $e'$  have concession at a reachable marking  $M$  and share a common precondition, so  ${}^\circ e \cap {}^\circ e' \neq \emptyset$ , the events  $e, e'$  are said to be in *conflict* at  $M$ . You can see why; if one occurs at  $M$  then the other does not. On the other hand, if  $M \xrightarrow{A} M'$  the events in  $A$  are said to occur *concurrently*.

So much for the objects of our first category. But what are the morphisms? To motivate these let’s look at some constructions that are often seen on nets—see [LC] for an early example. They arise when we consider how to give a Petri-net semantics to our generic programming language  $\text{Proc}_L$ , for a synchronisation algebra  $L$ . Of course when we model a term in  $\text{Proc}_L$  by a net, the events of the net will correspond to the actions of communication specified by the term, so the net carries a little extra structure, a labelling of events by elements of  $L$ . Fortunately we can factor the constructions on labelled nets into a construction on the nets proper and an extra construction due to the labelling.

Let us first define the guarding construction in  $\text{Proc}_L$ . The guarding construction  $\lambda p$  simply precedes the occurrences of events of the net  $N$  for  $p$  by an event of kind  $\lambda$ . A net construction which achieves this can be drawn in this way:

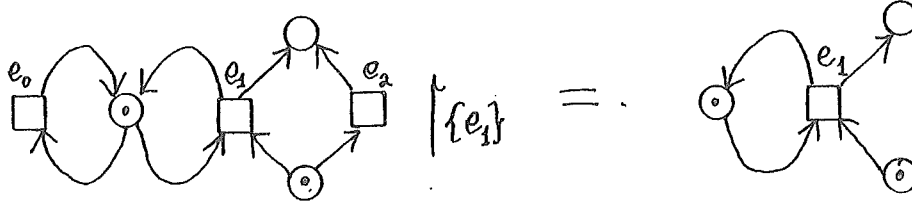


Simply adjoin a new event  $e$ , to form the new net  $eN$ , with the event  $e$  labelled by  $\lambda$ , to the net so that on its occurrence it sets-up the original initial marking.

Now we look at the restriction operation on nets. This simply disallows certain kinds of events from occurring. It can be modelled simply by “deleting” the forbidden events from the net.

**Restriction:** Let  $N = (B, E, F, M_0)$  be a net. Let  $E' \subseteq E$ . Define the restriction of  $N$  to  $E'$  to be  $N[E'] = (B, E', F', M_0)$  where  $F'$  is  $F$  restricted to  $(B \times E') \cup (E' \times B)$  i.e.  $F'_{b,e} = F_{b,e}$  and  $F'_{e,b} = F_{e,b}$  for  $e \in E'$  and  $b \in B$ .

**Example.** Here is a net with its restriction to a subset of events



The behaviour of a net restricted to a set of events is a restriction of the behaviour of the original net.

**Proposition.** Let  $N = (B, E, F, M_0)$  be a net. Let  $E' \subseteq E$ . Let  $M$  and  $M'$  be markings of  $N$ . Then

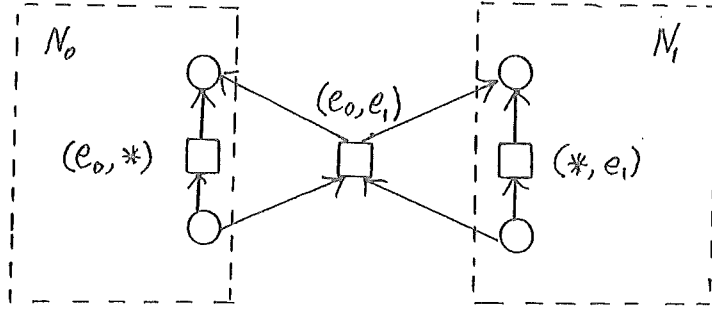
$$N[E'] : M \xrightarrow{A} M' \Leftrightarrow N : M \xrightarrow{A} M' \ \& \ A \in \mu E'.$$

Of course if a net is labelled, and so a structure of the form  $N = (B, E, F, M_0, l)$  with  $l : E \rightarrow L \setminus \{*, 0\}$ , and  $\Lambda \subseteq L$ , then we can define restriction to  $\Lambda$  to be the labelled net restricted to those events with labels in  $\Lambda$ . This models the operator  $[\Lambda$  in  $\text{Proc}_L$ .

**Product:** Imagine two processes, modelled as nets, set in parallel, side-by-side. Whether or not they communicate, to form events of synchronisation, depends on the what kinds of events they are prepared to do. This can be expressed by labelling the events by elements of a synchronisation algebra. The product of two (unlabelled) nets allows arbitrary synchronisations. When they are labelled, forbidden synchronisations can be removed by restriction.

Let  $N_0 = (B_0, E_0, F_0, M_0)$  and  $N_1 = (B_1, E_1, F_1, M_1)$  be nets. Rather than give a formal definition of their product—which can after all be found in [W3]—we describe the product construction in graphical terms. Disjoint copies of the two nets  $N_0$  and  $N_1$  are juxtaposed and extra events of synchronisation of the form  $(e_0, e_1)$  are adjoined, for  $e_0$  an event of  $N_0$  and  $e_1$  an event of  $N_1$ ; an extra event  $(e_0, e_1)$  has as preconditions those of its components so  ${}^\circ e = {}^\circ e_0 + {}^\circ e_1$  and similarly postconditions  $e_0^\circ + e_1^\circ$ . It is useful to think of the copies of the original events, those which are not synchronised with any companion event of the the other process as having the form  $(e_0, *)$  in the copy of  $N_0$  and the form  $(*, e_1)$  in the copy of  $N_1$ . Then the events of the product have the form  $E = \{(e_0, *) \mid e_0 \in E_0\} \cup \{(e_1, *) \mid e_1 \in E_1\} \cup \{(e_0, e_1) \mid e_0 \in E_0 \ \& \ e_1 \in E_1\}$ , which is the product of the sets  $E_0$  and  $E_1$  in the category of sets with partial functions. And similarly to be more precise about the conditions we can assume that they have the form  $B = B_0 \uplus B_1$  the disjoint union of  $B_0$  and  $B_1$ .

The product of  $N_0$  and  $N_1$ :



Write  $N_0 \times N_1$  for the product of the nets  $N_0$  and  $N_1$ .

So far all we have described, and informally at that, is a graphical construction on nets. To justify the construction we must understand the behaviour of the product of two nets in terms of the behaviour of the original nets. For this we need to project the behaviour of the product net to the behaviour of a component net. There are two parts to such a projection, an event part and a condition part. Consider the projection from  $N_0 \times N_1$  to  $N_0$ . There is an obvious partial function from the events of the product to the events of a component. Define  $\pi_0 : E \rightarrow E_0$  by  $\pi_0(e_0, e_1) = e_0$ —this will be undefined if  $e_0 = *$ . Define  $\rho_0$  to be the converse relation to the injection  $B_0 \rightarrow B$ . This projects conditions in the product back to the component—again it is a partial function. Now with the help of these two maps we can describe the behaviour of  $N_0 \times N_1$ .

**Proposition.** 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

$$N_0 \times N_1 : M \xrightarrow{A} M' \quad \text{iff} \quad (N_0 : \rho_0 M \xrightarrow{\pi_0 A} \rho_0 M' \ \& \ N_1 : \rho_1 M \xrightarrow{\pi_1 A} \rho_1 M').$$

A marking  $M$  is reachable in  $N_0 \times N_1$  iff  $\rho_0 M$  is reachable in  $N_0$  and  $\rho_1 M$  is reachable in  $N_1$ .

Intuitively the behaviour of the product is precisely that allowed when we project into the components. The pair of maps  $(\pi_0, \rho_0)$  specifies how the dynamic behaviour of the product of nets,  $N_0 \times N_1$ , projects to the dynamic behaviour in the component  $N_0$ . The pair  $(\pi_1, \rho_1)$  plays the same role but for the component  $N_1$ . They are essential in describing the behaviour of the product of nets.

**Parallel composition:** Let us look at the parallel composition of two nets. Assume now events of the nets  $N_0$  and  $N_1$  carry labels from a synchronisation algebra  $L$ . Let  $(N_0, l_0)$  and  $(N_1, l_1)$  be labelled nets, so  $l_i : E_0 \rightarrow L \setminus \{*, 0\}$  for  $i = 0, 1$ . Their parallel composition  $(N_0, l_0) \textcircled{D} (N_1, l_1)$  is a labelled net which is the restriction of the product to those events allowed by the synchronisation algebra  $L$  i.e.

$$\begin{aligned} (N_0, l_0) \textcircled{D} (N_1, l_1) &= ((N_0 \times N_1)[E', l]) \quad \text{where} \\ E' &= \{e \in E \mid l_0 \pi_0(e) \circ l_1 \pi_1(e) \neq 0\} \\ l(e) &= l_0 \pi_0(e) \circ l_1 \pi_1(e). \end{aligned}$$

Of course we want to understand the behaviour of the parallel composition of nets in terms of the behaviour of its components. But this follows from our observations about the restriction and product of nets.

**Proposition.** The behaviour of the parallel composition of nets  $(N_0, l_0)$  and  $(N_1, l_1)$  labelled by a synchronisation algebra is related to the behaviour of its components  $N_0$  and  $N_1$  by

$$N_0 \otimes N_1 : M \xrightarrow{A} M' \quad \text{iff} \quad A \in \mu\{e \in E \mid l_0\pi_0(e) \circ l_1\pi_1(e) \neq 0\} \ \& \\ N_0 : \rho_0 M \xrightarrow{\pi_0 A} \rho_0 M' \quad \& \quad N_1 : \rho_1 M \xrightarrow{\pi_1 A} \rho_1 M'.$$

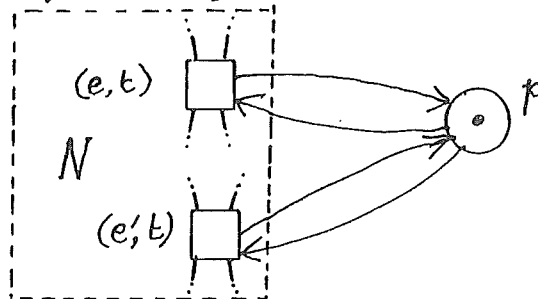
**Synchronous product:** Another important construction can be derived from the product construction with restriction, that of synchronous product. It is the restriction of the product of two nets to events of the form  $(e_0, e_1)$  where both  $e_0$  and  $e_1$  must be proper, non- $*$  events. Thus there is a tight synchronisation between the components of a synchronous product; in order to occur within a synchronous product every event of one component must synchronise with an event from the other.

Let  $N_0 = (B_0, E_0, F_0, M_0)$  and  $N_1 = (B_1, E_1, F_1, M_1)$  be nets. Define their synchronous product  $N_0 \otimes N_1$  to be the restriction  $N_0 \times N_1 \upharpoonright (E_0 \times E_1)$ . There are obvious projections got by restricting the projections of the product.

**Example.** One can represent a ticking clock as the following simple net, call it  $\Omega$ :



Given an arbitrary contact-free net  $N$  it is a simple matter to serialise, or interleave, its event occurrences; just synchronise them one at a time with the ticks of the clock. This amounts to forming the synchronous product  $N \otimes \Omega$  of  $N$  with  $\Omega$ , in a picture:



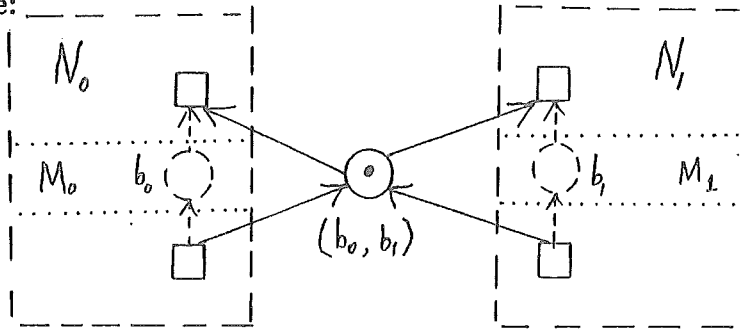
It is easy to check that the synchronous product  $N \otimes \Omega$  does serialise the event occurrences of  $N$ —just use the properties of restriction and product.

**Proposition.**  $M$  is a reachable marking of  $N \otimes \Omega$  and  $M \xrightarrow{A} M'$  in  $N \otimes \Omega$  iff  $M - p$  is a reachable marking of  $N$  and  $\exists e. A = (e, t) \ \& \ N : (M - p) \xrightarrow{e} (M' - p)$ .

**Sum:** We define the sum construction but only for contact-free nets as I'm not sure what the general construction should be. Roughly the sum construction fuses together the

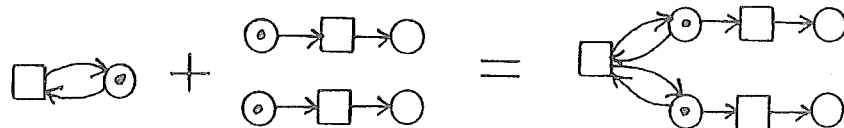
initial markings of two nets, so the resulting net either behaves like one component or the other. Again we shall describe the construction graphically—refer to [W3] for a formal definition.

Let  $N_0 = (B_0, E_0, F_0, M_0)$  and  $N_1 = (B_1, E_1, F_1, M_1)$  be contact-free nets. The two nets  $N_0$  and  $N_1$  are laid side by side and then a little surgery is performed on their initial markings. For each pair of conditions  $b_0$  in the initial marking of  $N_0$  and  $b_1$  in the initial marking of  $N_1$  a new condition  $(b_0, b_1)$  is created and made to have the same pre and post events as  $b_0$  and  $b_1$  together. The conditions in the original initial markings are removed and replaced by a new initial marking consisting of these newly created conditions. Here is the picture:



Notice a condition in the initial marking of one component is generally represented by more than one condition in the initial marking of the sum.

**Example.** The sum of two nets:



The set of events of the sum  $E$  is the disjoint union  $E_0 \uplus E_1$  of the events of the components. There are the obvious injections  $in_0 : E_0 \rightarrow E$  and  $in_1 : E_1 \rightarrow E$  on events. The initial marking of the sum can be represented by  $M = M_0 \times M_1$ , and its set of conditions by

$$B = \{(b_0, *) \mid b_0 \in B_0 \setminus M_0\} \cup \{(*, b_1) \mid b_1 \in B_1 \setminus M_1\} \cup M.$$

Then there are the obvious injection relations  $\iota_0$  and  $\iota_1$  where

$$\begin{aligned} b_0 \iota_0 b &\Leftrightarrow \exists b_1 \in B_1 \cup \{*\}. b = (b_0, b_1), \\ b_1 \iota_1 b &\Leftrightarrow \exists b_0 \in B_0 \cup \{*\}. b = (b_0, b_1). \end{aligned}$$

Thus the injection relations are opposite to the obvious partial functions taking a condition in  $B$  to its first or second component. Using the injections we can express the behaviour of the sum in terms of the behaviour of its components.

**Proposition.** Let  $N_0 + N_1$  be the sum of contact-free nets with injections  $(in_0, \iota_0)$  and  $(in_1, \iota_1)$ . Then  $M$  is a reachable marking of  $N_0 + N_1$  and  $M \xrightarrow{A} M'$  iff

$\exists$  reachable marking

$$M_0, A_0, M'_0. N_0 : M_0 \xrightarrow{A_0} M'_0 \ \&\& \ A = in_0 A_0 \ \&\& \ M = \iota_0 M_0 \ \&\& \ M' = \iota_0 M'_0$$

or

$\exists$  reachable marking

$$M_1, A_1, M'_1. N_1 : M_1 \xrightarrow{A_1} M'_1 \ \&\& \ A = in_1 A_1 \ \&\& \ M = \iota_1 M_1 \ \&\& \ M' = \iota_1 M'_1.$$

Those familiar with Milner's work may be a little bothered by our definition of sum. For the  $+$  of CCS and SCCS once a component has been selected nondeterministically the choice is stuck to, which is not true in general for our sum—consider the example above. However our construction will agree with Milner's on those contact-free nets for which  $\forall b \in M_0 \not\exists e. eFb$  i.e. no event leads into the initial marking. If one were to systematically give a net semantics to the languages  $\text{Proc}_L$ , which include CCS and SCCS, all the nets constructed would be contact-free and satisfy this property.

We do not describe the recursive definition of nets in detail here. Such nets can be defined in the standard way one builds-up sets by inductive definitions (see [Ac]); one must however take a little care to ensure that the operations on nets are monotonic with respect to the ordering of coordinatewise inclusion on nets, but this is not hard (see [W3], [Stu] or [GM] though the last is unnecessarily complicated). Alternatively, recursion can be handled in a categorical setting using the notion  $\omega$ -limits of chains of net-embeddings and  $\omega$ -continuous functors spelt-out in [W3]. We leave the other constructions for  $\text{Proc}_L$  to the reader.

Projections and injections on nets are examples of a more general notion of morphism between nets. It seems from our examples that a morphism from a net  $N$  to a net  $N'$  should express how the behaviour of  $N$  induces a behaviour in  $N'$ . We look for a general definition.

Given that we can regard a net as a 2-sorted algebra, an obvious first attempt is to take morphisms to be *homomorphisms* of the associated net-algebras. Because they are algebras over multisets the maps should be *linear*. Let's spell out what it means to be such a homomorphism.

Let  $N = (B, E, F, M)$  and  $N' = (B', E', F', M')$  be nets. A homomorphism from  $N$  to  $N'$  is a pair of multifunctions  $(\eta, \beta)$  with  $\eta : E \rightarrow_{\mu} E'$  and  $\beta : B \rightarrow_{\mu} B'$  such that

$$\beta M = M' \ \&\& \ \forall A \in \mu E. \circ(\eta A) = \beta(\circ A) \ \&\& \ (\eta A)^\circ = \beta(A^\circ).$$

You can see a homomorphism of nets preserves initial markings and the condition-environments of events. We are on the right track because homomorphisms do indeed preserve the dynamic behaviour of nets.



**Proposition.** Let  $(\eta, \beta) : N \rightarrow N'$  be a homomorphism of nets. If  $M \xrightarrow{A} M'$  in  $N$  then  $\beta M \xrightarrow{\eta A} \beta M'$  in  $N'$ .

If  $(\eta, \beta)$  is a homomorphism from  $N$  to  $N'$ , as a computation

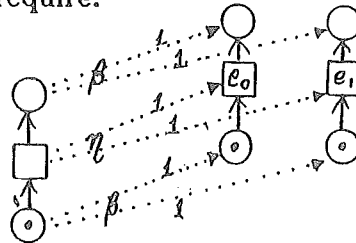
$$M_0 \xrightarrow{A_0} M_1 \xrightarrow{A_1} \dots \xrightarrow{A_{n-1}} M_n \xrightarrow{A_n} \dots$$

is traced-out in  $N$  so the computation

$$\beta M_0 \xrightarrow{\eta A_0} \beta M_1 \xrightarrow{\eta A_1} \dots \xrightarrow{\eta A_{n-1}} \beta M_n \xrightarrow{\eta A_n} \dots$$

is traced-out in  $N'$ . This is not to say that all homomorphisms on nets should be morphisms. But if the morphisms are homomorphisms they will automatically preserve behaviour, a property we certainly require.

**Example.** A homomorphism:



There is a problem with the interpretation of the homomorphism in this example. The occurrence of a single event in the domain of the homomorphism induces the simultaneous or coincident occurrence of 2 events  $e_0$  and  $e_1$  in its range. This goes against a view of Petri nets expressed by Petri that events which are coincident are the same event. Morphisms should be homomorphisms which preserve events, in the sense that  $\eta$  should be a partial function, thus forbidding the example above. Note we do not want morphisms to "preserve conditions" in the sense that  $\beta$  should be a partial function; to do so would rule out the injections used to characterise the behaviour of our sum construction.

**Definition.** A morphism of Petri nets is a homomorphism  $(\eta, \beta)$  in which  $\eta$  is a partial function. A morphism  $(\eta, \beta)$  is synchronous when  $\eta$  is a total function.

When nets are contact-free, just as their behaviour can be described using sets and relations instead of multisets and multifunctions, so can morphisms be characterised in a more elementary manner.

**Proposition.** Let  $N = (B_0, E_0, F_0, M_0)$  and  $N_1 = (B_1, E_1, F_1, M_1)$  be contact-free nets. A pair  $(\eta, \beta)$  is a morphism  $N_0 \rightarrow N_1$  iff  $\eta$  is a partial function, and  $\beta$  is a relation between  $B_0$  and  $B_1$  such that:

- (i)  $M_1 = \beta M_0$  and  $\forall b_1 \in M_1 \exists! b_0 \in M_0. b_0 \beta b_1$ ,
- (ii) If  $b_0 \beta b_1$  then

$\eta$  restricts to a total function  ${}^\circ b_0 \rightarrow {}^\circ b_1$  and

$\eta$  restricts to a total function  $b_0^\circ \rightarrow b_1^\circ$ ,

- (iii) If  $e_0 \eta e_1$  then

$\beta^{op}$  restricts to a total function  ${}^\circ e_1 \rightarrow {}^\circ e_0$  and

$\beta^{op}$  restricts to a total function  $e_1^\circ \rightarrow e_0^\circ$ .

(We use  $R^{op}$  for the opposite relation to  $R$ .)

**Definition.** Let  $\mathbf{Net}$  be the category of nets with net morphisms. Let  $\mathbf{Net}^c$  be the full subcategory of nets with objects just the contact-free nets. Let  $\mathbf{Net}_{syn}$  and  $\mathbf{Net}_{syn}^c$  be their subcategories in which morphisms are synchronous.

The constructions we have seen turn out to be categorical constructions. Recall the definition of product in a category. A product of two objects  $N_0$  and  $N_1$  consists of an object  $N_0 \times N_1$  with projection morphisms  $\Pi_0 : N_0 \times N_1 \rightarrow N_0$  and  $\Pi_1 : N_0 \times N_1 \rightarrow N_1$  which satisfy the property that given any pair of morphisms  $f_0 : N \rightarrow N_0$  and  $f_1 : N \rightarrow N_1$  there is a unique morphism  $[f_0, f_1] : N \rightarrow N_0 \times N_1$  such that  $f_0 = \Pi_0 \circ [f_0, f_1]$  and  $f_1 = \Pi_1 \circ [f_0, f_1]$ . Coproduct is the dual notion got by reversing the arrows. Categorical constructions such as these are unique to within isomorphism.

**Proposition.**

The product  $N_0 \times N_1$ , with morphisms  $(\pi_0, \rho_0)$  and  $(\pi_1, \rho_1)$ , is a categorical product in  $\mathbf{Net}$ , the category of nets.

The synchronous product  $N_0 \otimes N_1$ , with morphisms the restrictions of the projections is a product in  $\mathbf{Net}_{syn}$ , the category of nets with synchronous morphisms.

The sum  $N_0 + N_1$  with injections  $(in_0, \iota_0)$  and  $(in_1, \iota_1)$  is a coproduct in both the subcategories of contact-free nets with morphisms and synchronous morphisms.

The fact that parallel composition is so closely related to a product adds mathematical substance to the intuition, often expressed, that parallelism is some kind of "orthogonality".

So what? Well, one important consequence of the constructions being categorical is that each comes accompanied by a characterisation to within isomorphism. This means

that we need not worry about the details of the concrete and ad hoc construction we chose to build-up our product, synchronous product and sum. But more important perhaps is the use to which these facts can be put when we translate between different models. They too can be made into categories. In them too parallel compositions are obtained by restricting the product, and the sum of processes will be modelled as a coproduct. And it happens that the categories can be related by functors, passing back and forth, in such a way that the categorical constructions are preserved. We see a typical example of how in the next section.

We turn to consider other models, and see how they can be embedded in the category of nets. For the most part we shall be very sketchy in our treatment of labelled versions of these other models, but because we show how they can be identified with labelled nets it follows how to perform constructions on these other labelled structures.

### 3. Occurrence nets—the semantics of Petri nets.

Nets are rather complex objects with an intricate behaviour which so far has been expressed in a dynamic way. We would like to know when two nets have essentially the same behaviour. In this section we propose a more “static” representation of their behaviour as a certain kind of net, a net of condition and event occurrences. This is a generalisation of the familiar unfolding of a state–transition system to a tree [W2]. The theorems of this section only work if we restrict the class of nets and we will assume that the nets are contact–free. The occurrence net we associate with a contact–free net will be built–up essentially by unfolding the net to its occurrences. This unfolding is a canonical representative of the behaviour of the original net. Occurrence nets and the operation of unfolding a net to an occurrence net were first introduced in [NPW, W].

We want to axiomatise those nets in which conditions and events correspond to occurrences (as is the case with Petri’s causal nets but we also wish to represent conflict in these nets).

An occurrence net is a contact–free net  $(B, E, F, M)$  for which the following restrictions are satisfied:

- (i)  $b \in M \Leftrightarrow {}^\circ b = \emptyset$ , so the initial marking is identified with the set of conditions which are not preceded by any events in the  $F$ –relation,
- (ii)  $\forall b \in B. |{}^\circ b| \leq 1$ , so a condition can be caused to hold through the occurrence of at most one event,
- (iii)  $F^+$  is irreflexive and  $\forall e \in E. \{e' \mid e'F^*e\}$  is finite, so we ban repetitions of the same event and insist the occurrence of an event can only depend on the occurrence of a finite number of events,
- (iv)  $\#$  is irreflexive where

$$e\#_1 e' \Leftrightarrow_{def} e \in E \ \& \ e' \in E \ \& \ {}^\circ e \cap {}^\circ e' \neq \emptyset \ \text{and} \\ x\#x' \Leftrightarrow_{def} \exists e, e' \in E. e\#_1 e' \ \& \ eF^*x \ \& \ e'F^*x'.$$

In this way we eliminate those events which cannot possibly occur because they depend on the previous occurrence of conflicting events.

Suppose  $N = (B, E, F, M)$  is an occurrence net. We call the relation  $\#_1$  defined above the *immediate conflict relation* and  $\#$  the *conflict relation*. We define the *concurrency relation*,  $co$ , between pairs  $x, y \in B \cup E$  by:

$$x \text{ co } y \Leftrightarrow_{def} \neg(xF^+y \ \text{or} \ yF^+x \ \text{or} \ x\#y).$$

Being nets, occurrence nets determine a subcategory of  $\mathbf{Net}$ .

**Definition.** Write  $\text{Occ}$  for the category of occurrence nets with net morphisms. Write  $\text{Occ}_{\text{syn}}$  for the subcategory of occurrence nets with synchronous morphisms.

For occurrence nets there is an especially simple definition of a *concurrency relation* and *conflict relation* which was previously only defined with respect to a marking.

**Proposition.** Let  $N = (B, E, F, M)$  be an occurrence net. Then every event of  $N$  has concession at some reachable marking and every condition of  $N$  holds at some reachable marking.

Let  $e, e'$  be two events of  $N$ . Let  $b, b'$  be two conditions of  $N$ .

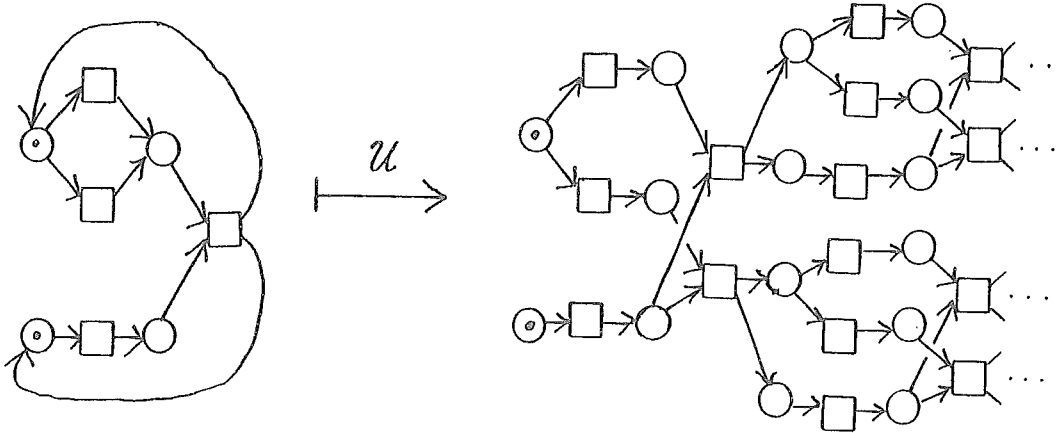
The relations  $\#_1 \subseteq E^2$  and  $\# \subseteq (B \cup E)^2$  are binary, symmetric, irreflexive relations. The relation of immediate conflict  $e \#_1 e'$  holds iff there is a reachable marking of  $N$  at which the events  $e$  and  $e'$  are in conflict.

The relation  $co$  is a binary, symmetric, reflexive relation between conditions and events of  $N$ . We have  $b co b'$  iff there is a reachable marking of  $N$  at which  $b$  and  $b'$  both hold. We have  $e co e'$  iff there is a reachable marking at which  $e$  and  $e'$  can occur concurrently.

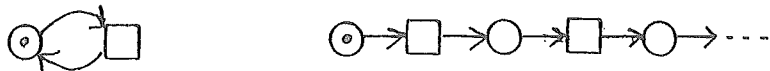
**Proposition.** Let  $N = (B, E, F, M)$  be a contact-free net. There is a unique occurrence net  $\mathcal{U}N = (B', E', F', M')$  with a folding  $f = (\eta, \beta) : \mathcal{U}N \rightarrow N$  which satisfies:

$$\begin{aligned}
 B' &= \{(\emptyset, b) \mid b \in M\} \cup \{(\{e_0\}, b) \mid e_0 \in E' \ \& \ b \in B \ \& \ \eta(e_0)Fb\}, \\
 E' &= \{(S, e) \mid S \subseteq B' \ \& \ e \in E \ \& \ \beta S = \circ e \ \& \ \forall b_0, b'_0 \in S. b_0 co b'_0\}, \\
 xF'y &\Leftrightarrow \exists w, z. y = (w, z) \ \& \ x \in z, \\
 M' &= \{(\emptyset, b) \mid b \in M\}, \\
 &\text{and} \\
 e_0 \eta e &\Leftrightarrow \exists S \subseteq B'. e_0 = (S, e), \\
 b_0 \beta b &\Leftrightarrow b \in M \ \& \ b_0 = (\emptyset, b) \ \text{or} \ \exists e_0 \in E'. b_0 = (\{e_0\}, b).
 \end{aligned}$$

**Example.** This example illustrates a contact-free net together with its occurrence net unfolding.



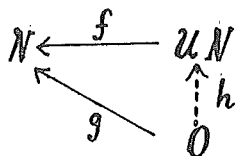
Something is lost in the passage from a net  $N$  to the occurrence net  $\mathcal{U}N$ . For example two non-isomorphic nets like



unfold to isomorphic occurrence nets. We have lost information about which events are occurrences of the same repetitive event. Still,  $\mathcal{U}N$  is in some sense the natural occurrence net which represents  $N$ . We make this precise.

The morphism  $f : \mathcal{U}N \rightarrow N$  expresses how the conditions and events in  $\mathcal{U}N$  are occurrences of conditions and events in the original contact-free net  $N$  where events and conditions may occur repeatedly. It is the construction  $\mathcal{U}N$  together with the morphism  $f$  which possess an abstract characterisation; the pair  $\mathcal{U}N, f$  is cofree over  $N$ . Informally this says that  $\mathcal{U}N$  is the “best” occurrence net to represent  $N$ . Formally it says

**Proposition.** *If  $O$  is any other occurrence net which maps  $g : O \rightarrow N$  then there is a unique morphism  $h : O \rightarrow \mathcal{U}N$  such that this diagram commutes:*



See [W3] for a proof.

Because this can be done for all contact-free nets  $N$  this implies by [Mac thm 2, p.81] that the operation of unfolding on nets extends to an operation on morphisms to make  $\mathcal{U}$  into a functor  $\text{Net}^c \rightarrow \text{Occ}$  which is the *right adjoint* to the inclusion functor  $\text{Occ} \rightarrow \text{Net}^c$ . Together the inclusion functor and  $\mathcal{U}$  determine an *adjunction* between  $\text{Occ}$  and  $\text{Net}^c$ ; the inclusion functor is the *left adjoint* and  $\mathcal{U}$  the *right adjoint* of the adjunction. This has some important consequences.

By [Mac thm 1, p. 114] right adjoints preserve limits and in particular products. Thus we know that

$$\mathcal{U}(N_0 \times N_1) \cong \mathcal{U}N_0 \times_{\text{Occ}} \mathcal{U}N_1,$$

for contact-free nets  $N_0$  and  $N_1$ , *i.e.* that if we take the product of two contact-free nets, and then unfold the result, we obtain the same net to within isomorphism as if we unfold the nets first, and then form their product in the category  $\text{Occ}$ .

The unfolding of an occurrence net is isomorphic to the occurrence net itself. Formally the morphism  $f_O : \mathcal{U}O \rightarrow O$  is a *natural isomorphism* for any occurrence net  $O$ . This makes the adjunction a bit special; it is a *coreflection* and  $\text{Occ}$  is a *coreflective subcategory* of  $\text{Net}^c$ . Thus we see that the product of two occurrence nets  $O_0$  and  $O_1$  in  $\text{Occ}$  is

$$O_0 \times_{\text{Occ}} O_1 \cong \mathcal{U}O_0 \times_{\text{Occ}} \mathcal{U}O_1 \cong \mathcal{U}(O_0 \times_{\text{Net}} O_1).$$



Because  $\mathcal{U}$  is a right adjoint to the inclusion functor  $\text{Occ} \rightarrow \text{Net}^c$ , the inclusion functor is a left adjoint to  $\mathcal{U}$ , and so the inclusion functor preserves colimits like coproduct. This fact tells us that coproducts in  $\text{Occ}$  are the same, to within isomorphism, as coproducts in  $\text{Net}^c$  i.e.

$$O_0 +_{\text{Occ}} O_1 \cong O_0 +_{\text{Net}^c} O_1,$$

the coproduct in  $\text{Net}^c$ .

Proving these facts directly from the unfolding construction is quite unwieldy—and completely uninformative—so it is fortunate there is this abstract characterisation of the occurrence net unfolding of a contact-free net. In a sense it was there all the time, because the unfolding operation acts on nets as the right adjoint to the inclusion functor  $\text{Occ} \rightarrow \text{Net}^c$  so it was determined, to within natural isomorphism, by the categorical set-up.

Right adjoints preserve limits; they do not necessarily preserve colimits, and indeed here is a simple example where a coproduct is not preserved by unfolding.

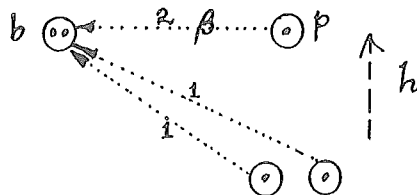
**Example.** This example is essentially the same as that given in [W2] for a category of transition systems where unfolding yields a tree. Let  $N$  be the net . Let  $\Omega$  be the net representing a clock, as seen before. Then  $\mathcal{U}(N + \Omega) \not\cong \mathcal{U}N + \mathcal{U}\Omega$ . 

Of course we can restrict to subcategories of nets so that unfolding does preserve coproducts. A subcategory, which have already remarked on, for which this is true is that where the nets satisfy: every condition in the initial marking has no pre-events.

The coreflection between  $\text{Occ}$  and  $\text{Net}^c$  cuts down to a coreflection between the subcategories of synchronous morphisms with the analogous results.

It is now a simple exercise to define operations on labelled occurrence nets which correspond to constructs of  $\text{Proc}_L$ , in the same way as we did for nets. For example, the parallel composition of labelled occurrence nets is the restriction of their product in  $\text{Occ}$ .

Why did we restrict ourselves to contact-free nets in this section? Although there does seem to be a natural occurrence-net unfolding of the more general nets it is easy to see that this cannot be cofree over the original net. Let the net  $N$  consist of a single condition  $b$  with the initial marking  $2b$ . Suppose there were an occurrence net  $O$  which was cofree over  $N$ . The occurrence net would *either* have an initial marking containing two conditions  $c, d$  so that in the “folding”  $f = (\beta, \eta) : O \rightarrow N$  we had  $\beta_{d,b} = \beta_{c,b} = 1$  or a single condition  $p$  such that  $\beta_{p,b} = 2$ . The latter case is impossible as we would then not have a morphism  $h$  making this diagram commute:



So if anything  $\beta_{c,b} = \beta_{d,b} = 1$  for  $c, d$  in the initial marking of  $O$ . But then of course we do not have a *unique* morphism making this diagram commute, as is shown by these two morphisms:



Thus we cannot have an occurrence net and morphism which is cofree over  $N$ . This argument works just as well to show there cannot in general be a contact-free net which is cofree over an arbitrary Petri net. The relationship between the categories  $\mathbf{Net}$  and  $\mathbf{Net}^c$ , and the categories  $\mathbf{Net}$  and  $\mathbf{Occ}$  does not seem to be as pleasant as that between the other categories we discuss here, and certainly is not understood as well.



#### 4. Event structures.

We show the relationship between a category of event structures and the category of nets. The event structures are of the simple form introduced in [NPW]; they consist of a set of events related by a causal dependency relation and a conflict relation. The paper [NPW] and thesis [W] contain a great deal to motivate event structures. The morphisms on event structures were introduced in [W1] and we refer the reader there and to [W] for the relationship between the event structures used here and more general event structures.

We show that constructions given in [NPW] determine an adjunction between nets and event structures. There is a right adjoint, part of a coreflection, from occurrence nets to event structures. This composes with the right adjoint of unfolding, part of a coreflection, to give a right adjoint from nets to event structures, consequently part of a coreflection. This pleasant categorical set-up makes it easy to relate semantics given in terms of nets to those in terms of event structures, and through them to the pomset model of Vaughan Pratt [Pr] and the *behaviour systems* of Mike Shields [Sh1,2].

An event structure can be regarded as an occurrence net with its conditions stripped away leaving the  $F^*$  relation as the causal dependency relation and the  $\#$  relation as conflict.

An event structure is a triple  $(E, \leq, \#)$  consisting of

- (i)  $E$  a set of events,
- (ii)  $\leq$  the causal dependency relation a partial order on  $E$  and
- (iii)  $\#$  the conflict relation a binary symmetric relation on  $E$

which satisfy  $e \# e' \leq e'' \Rightarrow e \# e''$  and  $[e] =_{def} \{e' \in E \mid e' \leq e\}$  is finite.

**Remark.** Notice that here we insist that event structures satisfy a finiteness restriction,  $[e] < \infty$ , a restriction not enforced in [NPW]. Event structures of the simple form above are called *prime event structures* in [W1, 2].

Event structures are accompanied by a natural idea of *configuration* (or state), the downwards-closed and conflict-free subsets of events with respect to  $\leq$  and  $\#$ . Intuitively a configuration is a set of events that occur in some history of a process; it should only be possible for an event to occur once the events on which it causally depends have occurred and it should be impossible for two events in conflict to occur in the same history.

Let  $(E, \leq, \#)$  be an event structure. Let  $x \subseteq E$ . Say  $x$  is *downwards-closed* iff  $\forall e, e' \in E. e \leq e' \in x \Rightarrow e \in x$ . Say  $x$  is *conflict-free* iff  $\forall e, e' \in x. \neg(e \# e')$ . Write  $\mathcal{L}(E, \leq, \#)$  for the set of left-closed conflict-free subsets.

The set of configurations of an event structure determine a *behaviour system* in the sense of [Sh1,2]. Imagine every event of an event structure labelled by an element of a

synchronisation algebra, to form a labelled event structure. Then each configuration is a pomset and the set of configurations is a process in the sense of [Pr]. This indicates the relationship between labelled event structures and the pomset model.

Clearly an occurrence net determines an event structure [NPW]; just strip the conditions away but remember the more abstract causal dependency and conflict relation they induce.

**Definition.** Let  $N = (B, E, F, M)$  be an occurrence net. Define

$$\mathcal{E}(N) = (E, F^* [E, \# [E).$$

A morphism  $(\eta, \beta) : N \rightarrow N'$  between occurrence nets  $N$  and  $N'$  consists in part of a partial function  $\eta : E \rightarrow E'$  between the associated sets of events. The partial function  $\eta$  always respects the event structures associated with the nets, in this sense:

$$\forall x \in \mathcal{L}(\mathcal{E}N). (\eta x \in \mathcal{L}(\mathcal{E}N') \ \& \ (\forall e, e' \in x. \eta(e) = \eta(e') \neq * \Rightarrow e = e')).$$

So, on the event structures the map  $\eta$  preserves configurations and the nature of events. It seems natural to take this as a definition of morphisms on event structures, so a morphism on event structures is a partial function between the sets of events which satisfies the property above.

**Definition.** Define  $\mathbb{P}$  to be the category of event structures obtained by taking morphisms on event structures to be partial functions on the sets of events which preserve configurations and events in the sense above; morphisms are composed as partial functions. Define  $\mathbb{P}_{syn}$  to be the subcategory of synchronous morphisms, in which maps are total.

Clearly  $\mathcal{E}$  extends to a functor  $\text{Occ} \rightarrow \mathbb{P}$  from occurrence nets to event structures by defining  $\mathcal{E}$  on morphisms  $(\eta, \beta)$  by

$$\mathcal{E}(\eta, \beta) = \eta.$$

Note we not only have a functor  $\mathcal{E} : \text{Occ} \rightarrow \mathbb{P}$  from occurrence nets to event structures but also the functor  $\mathcal{E} \circ \mathcal{U} : \text{Net}^c \rightarrow \mathbb{P}$ , translating arbitrary contact-free nets to event structures.

It is natural to ask if, conversely, an event structure can be identified with an occurrence net. Of course we would like every morphism between event structures to correspond to net morphism between the associated nets. We seek a functor  $\mathcal{N} : \mathbb{P} \rightarrow \text{Occ}$  which "embeds" the category of event structures in the category of occurrence nets, so  $\mathcal{E} \mathcal{N} E$  is naturally isomorphic to the original event structure  $E$ . Ideally, we would hope that  $\mathcal{E}$  would be a right adjoint to  $\mathcal{N}$  making a coreflection. This is indeed the case and we have

all the benefits explained in the last section. We explain the construction of  $\mathcal{N}$ , a minor modification of that in [NPW].

An event structure can be identified with a canonical occurrence net. The basic idea is to produce an occurrence net with as many conditions as are consistent with the causal dependency and conflict relations of the event structure. But we do not want more than one condition with the same beginning and ending events—we want an occurrence net which is “condition-extensional” in the terms of [Br]. Thus we can identify the conditions with pairs of the form  $(e, A)$  where  $e$  is an event and  $A$  is a subset of events causally dependant on  $e$  and with every distinct pair of events in  $A$  in conflict. But not quite, we also want initial conditions with no beginning events.

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

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

This time it is easier to establish the coreflection by showing the freeness of the occurrence net associated with an event structure.

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

Then  $\mathcal{N}(E, \leq, \#)$  is an occurrence net. Moreover,  $\mathcal{E} \circ \mathcal{N}(E, \leq, \#) = (E, \leq, \#)$ .

The net  $\mathcal{N}E$  and identity function  $1_E : E \rightarrow \mathcal{E}\mathcal{N}E$  is free over  $E$  with respect to  $\mathcal{E}$  i.e. for any morphism  $f : E \rightarrow \mathcal{E}N$  in  $\mathbb{P}$  there is a unique morphism  $h : \mathcal{N}E \rightarrow N$  in  $\text{Occ}$  such that  $\mathcal{E}h \circ 1_E = f$  (i.e.  $\mathcal{E}h = f$ ).

Thus there is a coreflection between event structures and occurrence nets with  $\mathcal{E}$  as its right adjoint and  $\mathcal{N}$  as its left adjoint. This composes with the coreflection between occurrence nets and contact-free nets we saw in the last section to give a coreflection between event structures and contact-free nets.

Reasoning in the same way as we did for the coreflection between  $\text{Net}^c$  and  $\text{Occ}$ , we see, for instance,

$$\begin{aligned} \mathcal{E}(N_0 \times_{\text{Occ}} N_1) &\cong \mathcal{E}N_0 \times_{\mathbb{P}} \mathcal{E}N_1 \\ E_0 \times_{\mathbb{P}} E_1 &\cong \mathcal{E}\mathcal{U}(\mathcal{N}E_0 \times_{\text{Net}} \mathcal{N}E_1) \\ E_0 +_{\mathbb{P}} E_1 &\cong \mathcal{E}\mathcal{U}(\mathcal{N}E_0 +_{\text{Net}} \mathcal{N}E_1), \end{aligned}$$

which translates constructions in one category to constructions in the other, giving the product and coproduct in  $\mathbb{P}$  in terms of the product and coproduct in  $\text{Net}^c$ .

With extra labelling structure one can carry out the construction for parallel composition pretty much as for nets. More direct definitions of the product, sum and parallel composition of event structures can be found in [W1].

## 5. Trees and synchronisation trees.

Trees underlie most interleaving models of parallel computation. The nodes represent states and the arcs occurrences of events. When the arcs are labelled by elements of a synchronisation algebra they are generally called *synchronisation trees*, a term introduced by Milner in [M1] for the special case when the synchronisation algebra is that for CCS. Trees can be identified with a special kind of event structure, and so of course with a special kind of net, just applying the results of the previous section.

A tree is an event structure in which any two compatible configurations are comparable. In other words, if, when ordered by inclusion, the partial order of the configurations of an event structure form a tree with limit points, we call the event structure a tree. This picks out those event structures  $(E, \leq, \#)$  in which pairs of events are either in conflict or related by causal dependency *i.e.*

$$\forall e, e' \in E. e \leq e' \text{ or } e' \leq e \text{ or } e \# e'.$$

You can check that in this representation of trees, finite configurations correspond nodes and events to arcs of a tree. We can pick out arcs of the tree by the "covering" relation on finite configurations

$$x \longrightarrow y \Leftrightarrow \exists e. e \notin x \ \& \ y = x \cup \{e\}.$$

As event structures, trees inherit a notion of morphism from the category  $\mathbf{P}$ . Let  $S$  and  $T$  be trees with finite configurations  $S^0$  and  $T^0$ , to be thought of as nodes. A morphism  $\theta : S \rightarrow T$  on trees corresponds to a map  $\hat{\theta} : S^0 \rightarrow T^0$  on nodes which satisfies these properties which can be easily understood in terms of the more familiar graphical view of trees:

- (i)  $\hat{\theta}(\emptyset) = \emptyset$  *i.e.* the root-node is preserved,
- (ii)  $x \longrightarrow y \Rightarrow \hat{\theta}(x) = \hat{\theta}(y)$  or  $\hat{\theta}(x) \longrightarrow \hat{\theta}(y)$  *i.e.* either arcs are preserved or collapsed.

Synchronous morphisms  $\theta$  correspond to those in which (ii) is replaced by the stronger property

$$x \longrightarrow y \Rightarrow \hat{\theta}(x) \longrightarrow \hat{\theta}(y).$$

**Definition.** Write  $\mathbf{Tr}$  for the category of trees and  $\mathbf{Tr}_{syn}$  for the subcategory with synchronous morphisms.

Again, as you'd expect by now, trees form a coreflective subcategory of event structures and this can be got by cutting down a coreflection between the category of contact-free nets  $\mathbf{Net}^c$  and  $\mathbf{Tr}$ . We need not look very far for the coreflection. Putting a contact-free

net  $N$  in synchronous product with the "ticking clock"  $\Omega$ , of section 2, we obtain the net  $N \otimes_{Net} \Omega$  in which the event occurrences are serialised. Unfolding this to an occurrence net and then taking the event structure associated with this we obtain a tree,

$$\mathcal{T}N = \mathcal{E}U(N \otimes \Omega).$$

Using the property that right adjoints preserve products we derive

$$\begin{aligned} \mathcal{T}N &= \mathcal{E}U(N \otimes \Omega) \\ &\cong (\mathcal{E}UN) \otimes_P (\mathcal{E}U\Omega) \\ &\cong (\mathcal{E}UN) \otimes_P \Omega_P \end{aligned}$$

where  $\Omega_P$  is the event structure associated with the "clock"; it consists of events  $0 \leq 1 \leq 2 \leq \dots \leq i \leq \dots$ . Thus  $\mathcal{T}N$  puts the event structure associated with  $N$  in synchronous product with the event-structure model of a clock. While intuitive it does require a proof that  $\mathcal{T}N$  is a tree. We refer to [W1] where it is shown that  $(- \otimes_P \Omega_P) : \mathcal{P} \rightarrow \mathcal{Tr}$  extends to a right adjoint of the inclusion functor  $\mathcal{Tr} \rightarrow \mathcal{P}$ , identifying trees with a certain kind of event structure. This adjunction is a coreflection. Composing it with the coreflection between contact-free nets and event structures we obtain

**Proposition.**  $\mathcal{T}$  extends to a functor which is right adjoint to the functor  $\mathcal{N} : \mathcal{Tr} \rightarrow \mathcal{Net}^c$ . They determine a coreflection between  $\mathcal{Tr}$  and  $\mathcal{Net}^c$ .

Thus there is an interleaving, or serialising, functor  $\mathcal{T} = \mathcal{E}U(- \otimes \Omega) : \mathcal{Net}^c \rightarrow \mathcal{Tr}$  which translates the non-interleaving models of Petri nets and event structures into the interleaving model of trees. We know that products are preserved by  $\mathcal{T}$  and that coproducts of trees coincide with their coproducts regarded as contact-free nets. Consequently, when we label nets and trees by elements of a synchronisation algebra we have that parallel compositions are preserved by interleaving and that the notion of sum agrees on trees whether we look on it as based on the coproduct of trees or the coproduct of trees regarded as nets.

By carrying out the construction in  $\mathcal{Net}^c$ , it is easy to see that the coproduct of trees has the effect of glueing them together at the root. The product has a more interesting characterisation, familiar from the work of Milner on synchronisation trees. Write  $\Sigma_{i \in I} T_i$  for the more general coproduct of a set of trees  $T_i$  indexed by  $i \in I$ ; this construction glues the set of trees together at their roots. The guarding operation  $eN$  on a net  $N$  gives a guarding operation on trees; it simply prefixes a tree  $T$  by an event  $e$ . Note any tree can be represented to within isomorphism as a coproduct of guarded trees  $\Sigma_{a \in A} aT_a$  for some set of events  $A$ . It is shown in [W1,2] that the product in the category of trees has the following form:

**Proposition.** *Let  $S$  and  $T$  be trees. Then*

$$S \cong \sum_{a \in A} aS_a \quad \text{and} \quad T \cong \sum_{b \in B} bT_b$$

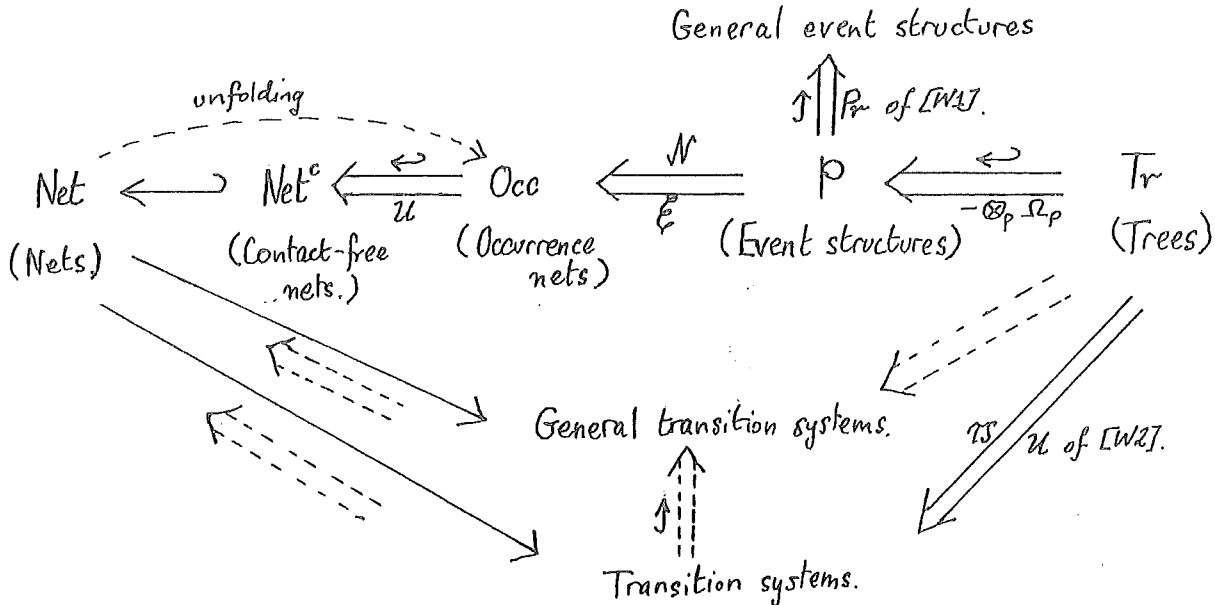
*for some sets of events  $A$  and  $B$  and trees  $S_a$  and  $T_b$  indexed by  $a \in A$  and  $b \in B$  respectively. We have the following characterisation of the product of  $S$  and  $T$  in  $\mathbf{Tr}$  :*

$$S \times T \cong \sum_{a \in A} (a, *)S_a \times T + \sum_{a \in A, b \in B} (a, b)S_a \times T_b + \sum_{b \in B} (*, b)S \times T_b.$$

Restricting the events of the product in accord with *e.g.* the synchronisation algebra for CCS we obtain the recursive characterisation of the parallel composition of synchronisation trees that Milner uses in [M1]. This reassuring fact demonstrates the coherence of the categorical view of constructs in a range of models.

## 6. Conclusion, loose ends.

There is a criss-cross of coreflections bridging different categories. We have seen some in this paper. I'll summarise those categories of models that have been related by coreflections in a diagram. (A diagram of the same pattern is valid for the synchronous subcategories, in which the morphisms are restricted to being synchronous.)



Coreflections are represented by double arrows in the direction of the left adjoint, and single functors by single arrow. Coreflections which are suspected, but not yet worked out are represented by dotted double arrows. As was pointed out there are fundamental obstacles to obtaining an adjunction between general nets and contact-free nets. Still, there does seem to be an obvious unfolding of a general net to an occurrence net. At present I have no idea of its categorical significance. And what about a coreflection between transition systems and nets? A category of transition systems is defined in [W2] and there is an obvious functor from nets in general to transition systems—take markings as states and single occurrences of events as transitions. If there is a left adjoint to that functor—there probably is—it will have to extend that which embeds trees in nets; the “obvious” identification of transition systems with nets in which states are represented as single conditions in a contact-free net cannot be the right adjoint. There is another variety of transition system, those in which transitions correspond to concurrent firings of a set of events—I’ve called them *general transition systems* in the diagram; I haven’t thought much about that category, though from our experience with nets it is clear how to define morphisms on it. The statement that net morphisms preserve the dynamic behaviour of nets translates into the fact that there is a functor from nets to the category of general transition systems, which can be taken to stand for net behaviour.

All the categories of models we have considered have been unlabelled. Constructions

on labelled objects were derived, though it was never defined what it meant to be a morphism between labelled nets for example. This can be done following the lines of [W2] in which categories of labelled trees (synchronisation trees) and transition systems are introduced. The idea is to restrict the morphisms further in accord with the extra labelling structure; the label of the image of an event should *divide* the label of the original event.

Each category has its own natural notion of equivalence, induced by isomorphism. We started with the Petri net model, our most concrete and least abstract model, with all its detail, and passed to the far abstract model of trees in which the concurrency structure is lost entirely. As far as the synchronisation-tree model is concerned the two CCS terms  $(\alpha nil | \beta nil)$  and  $(\alpha \beta nil + \beta \alpha nil)$  are the same—they denote isomorphic synchronisation trees—while as nets or event structures they are distinguished because they are not isomorphic.

There are, of course, many more abstract models yet, *e.g.* powerdomain models, models based on observational equivalence, the failure-set model and traces model, and their relationship is being understood better all the time. It remains to be clarified whether or not the categorical approach here can be generalised in a useful way to more abstract models. (The paper [LP] is an attempt, though I'm not convinced that the approach is useful for relating models, or is anything like general enough.) To do so it seems necessary to pass beyond the vocabulary of events and conditions or physical states to properties of the kind captured by temporal or modal logics, and take morphisms which respect these properties. The papers [S, Sm, Ab, LW, W4] portray denotational semantics in this light and to some extent the role of the categories here is taken by domains and the role of coreflections by embedding-projection pairs (see [W5]).

There is an apparent mismatch between the approach used here, with processes represented as objects in a category, and that generally used in denotational semantics where processes are taken to be elements of a domain, itself in a category of domains. Indeed it is hard to see how all the detailed structure of Petri nets, for example, could be caught adequately in a domain. It appears that sometimes we need a finer categorical structure than is possessed by a domain. This issue has appeared in another context, the work of Lehmann and Abramsky on generalisation of domains to categories [Ab1, Le].



## Appendix: multisets and multifunctions.

### Multisets

Let  $X$  be a set.

A *multiset* of  $X$  is a function  $f : X \rightarrow \omega$ . Write  $f_x$  for the *multiplicity*  $f(x)$  of the element  $x$ . Write  $\mu X$  for the set of multisets of  $X$ .

Let  $n \in \omega$ . Define  $\underline{n}$  of  $X$  to be the multiset  $\underline{n} : x \mapsto n$ . In particular, the *null multiset*  $\underline{0}$  of  $X$  is the function.  $\underline{0} : x \mapsto 0$ .

Let  $x \in X$ . Define the *singleton* multiset  $\hat{x}$  to be the function  $\hat{x} : y \mapsto \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise.} \end{cases}$  Say a multiset is a singleton if it has this form. Whenever it is clear from the context we shall write  $x$  for  $\hat{x}$ .

By convention, we shall identify subsets of  $X$  with those multisets of  $f \in \mu X$  such that  $f \leq \underline{1}$ .

### Operations on multisets

Many operations and relations on multisets are induced pointwise by operations and relations on integers.

Let  $f, g \in \mu X$ . Define

$$\begin{aligned} (f + g)_x &= f_x + g_x, \\ (f - g)_x &= \begin{cases} f_x - g_x & \text{if } f_x \leq g_x \\ 0 & \text{otherwise} \end{cases} \\ (f \vee g)_x &= \max\{f_x, g_x\} \\ (f \wedge g)_x &= \min\{f_x, g_x\} \end{aligned}$$

for  $x \in X$ . Define

$$f \leq g \Leftrightarrow \forall x \in X. f_x \leq g_x.$$

Occasionally it might happen that for example  $f$  is a multiset over a set  $X$  while  $g$  is a multiset over another set  $Y$ . In such a case we can still make sense of the operations above by simply extending  $f$  and  $g$  to be multisets over  $X \cup Y$ .

Let  $n \in \omega$  and  $f \in \mu X$ . Define their *scalar multiplication*  $nf$  to be the multiset given by  $(nf)_x = nf_x$  for  $x \in X$ .

### Multifunctions

Let  $X$  and  $Y$  be sets. A *multifunction* from  $X$  to  $Y$  is a function  $\theta : \mu X \rightarrow \mu Y$  which is *linear i.e.*

$$\theta(nf + mg) = n(\theta f) + m(\theta g)$$

for all  $n, m \in \omega$  and  $f, g \in \mu X$ . Write  $\theta : X \rightarrow_{\mu} Y$  when  $\theta$  is such a multifunction. Clearly the multifunction  $\theta$  determines and is determined by the matrix  $\theta_{x,y} = (\theta x)_y$  for  $x \in X, y \in Y$ , a multiset of  $X \times Y$ , and we shall often define a multifunction by giving its matrix.

By convention, we shall identify the relations between a set  $X$  and a set  $Y$  with those multifunctions  $\theta : X \rightarrow_{\mu} Y$  for which  $\theta_{x,y} \leq 1$ . In particular, we shall identify functions and partial functions with their extensions to multifunctions. We use  $*$  as a value to represent when the function is undefined. We shall use standard notation for relations and functions e.g. writing  $xRy$  when  $x$  and  $y$  are in relation  $R$ . For a relation  $R$  we use  $R^{op}$  to represent the converse or opposite relation  $xR^{op}y \Leftrightarrow_{def} yRx$ .

## Acknowledgements

I have learnt much from discussions with Mogens Nielsen and Gordon Plotkin, and from working with students on their projects at Aarhus University. I saw the definition of morphisms on general nets, and not just the contact-free ones, in discussion with Ursula Goltz. The notion of folding used by Ursula and Wolfgang Reisig in [GR] is a special kind of morphism as used in Net.

## References

[Ab] Abramsky, S., Domain theory as a theory of experiments. In the proceedings of the joint UK and USA seminar on concurrency, held at Carnegie-Mellon University, Pittsburgh July 1984, to appear as a volume of the Springer Lecture Notes in Comp. Sc. (1985). \*

[Ab1] Abramsky, S., Ph. D. thesis, University of London. In preparation.

[Ac] Aczel, P., An introduction to inductive definitions. In the handbook of Mathematical Logic, Ed. Barwise, J., North-Holland (1983).

[AM] Arbib, M.A., and Manes, E.G., Arrows, Structures and Functors, The categorical imperative. Academic Press (1975).

[B] Brookes, S.D., On the relationship of CCS and CSP. ICALP 1983, in Springer-Verlag Lecture Notes in Comp. Sc., vol.154 (1984).

[Br] Brauer, W.(Ed.), Net Theory and Applications, Springer-Verlag Lecture Notes in Comp. Sci., vol.84 (1980).

[CH] Campbell, R. H., and Habermann, A. N., The Specification of Process Synchronisation by Path Expressions. Springer-Verlag Lecture Notes in Comp. Sc. Vol.16 (1974).

[GR] Goltz, U. and Reisig, W., Processes of Place/Transition Nets. Icalp 83 and to appear in Information and Control.

[GM] Goltz, U. and Mycroft, A., On the relationship of CCS and Petri nets. ICALP 84 Springer-Verlag Lecture Notes in Comp. Sc. Vol.172 (1984).

[H] Hoare, C.A.R., Communicating sequential processes. Comm. ACM 21 (1978).

[HBR] Hoare, C.A.R., Brookes, S.D., and Roscoe, A.W., A Theory of Communicating Processes, Technical Report PRG-16, Programming Research Group, University of Oxford (1981); in JACM (1984).

[Le] Lehmann, D., Categories for fixed-point semantics. FOCS 17 (1976).

[LC] Lauer, P. E. and Campbell, R. H., Formal semantics for a class of high-level primitives for coordinating concurrent processes. *Acta Informatica* 5 pp.297-332 (1974).

[LP] Labella, A., and Peterossi, A., Towards a Categorical Understanding of Parallelism. Report of Istituto di Analisi dei Sistemi ed Informatica del C.N.R., Rome (1983).

[LW] Larsen, K. and Winskel, G., Using Information Systems to solve Recursive Domain Equations Effectively. *Springer Lecture Notes in Comp. Sc.*, vol. 173 (1984).

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

[M1] Milner, R., *A Calculus of Communicating Systems*. Springer Lecture Notes in Comp. Sc. vol. 92 (1980).

[M2] Milner, R., Calculi for synchrony and asynchrony, *Theoretical Computer Science*, pp.267-310 (1983).

[Mi] Milne, G., CIRCAL and the representation of communication, concurrency and time. Report of Comp Sc Dept, University of Edinburgh (1983).

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

[Pr] Pratt, V. R., On the composition of processes. ACM, Stanford University (1982), and see the paper in the proceedings\* (see [Ab]).

[Pe] Peterson, J. L., *Petri Net Theory and the Modelling of Systems*. Prentice-Hall (1981).

[R] Reisig, W., Petri nets. *Springer Lecture Notes in Comp. Sc.*, to appear.

[S] Scott, D., Domains for Denotational Semantics, *Springer-Verlag Lecture Notes in Comp. Sc.* 140 (1982).

[Sh1,2] Shields, M., Non-sequential behaviours: 1 and 2. Reports of the Comp. Sc. Dept., University of Edinburgh (part 1: 1982, part 2: 1983).

[Stu] Student projects for the course "Models for Concurrency", Computer Science Dept., University of Aarhus, Denmark (1980).

[Sm] Smyth, M.B., Power domains and predicate transformers: a topological view. *Proc. of ICALP 83*, Springer Lecture Notes in Comp. Sc. vol. 154 (1983).

[W] Winskel, G., Events in Computation. Ph.D. thesis, University of Edinburgh (1980).

[W1] Winskel, G., Event structure semantics of CCS and related languages, Springer-Verlag Lecture Notes in Comp. Sc. 140 and as a report of the Computer Sc. Dept., University of Aarhus, Denmark (1982).

[W2] Winskel, G., Synchronisation trees. Technical Report, Comp. Sc. Dept., Carnegie-Mellon University (1983). To appear in Theoretical Computer Science.

[W3] Winskel, G., A New Definition of Morphism on Petri Nets. Springer Lecture Notes in Comp Sc, vol. 166 and also as a report of the Computer Laboratory, University of Cambridge (1984).

[W4] Winskel, G., A note on powerdomains and modality. Springer Lecture Notes in Comp Sc, vol. 158 (1984).

[W5] Winskel, G., A complete proof system for SCCS with modal assertions. In preparation.