To appear in: Lecture Notes in Computer Science,
Proceedings PARLE Conference, Eindhoven, 1987.

Petri Net Models for Algebraic Theories of Concurrency

(extended abstract)

Rob van Glabbeek and Frits Vaandrager
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In this paper we discuss the issue of interleaving semantics versus True concurrency in an algebraic setting. We present various equivalence notions on Petri nets which can be used in the construction of algebraic models:
(a) the occurrence net equivalence of Nielsen, Plotkin & Winskel;
(b) bisimulation equivalence, which leads to a model which is isomorphic to the graph model of Baeten, Bergstra & Klop;
(c) the concurrent bisimulation equivalence, which is also described by Nielsen & Thiagarajan, and Goltz;
(d) partial order equivalences which are inspired by work of Pratt, and Boudol & Castellani.
A central role in the paper will be played by the notion of real-time consistency. We show that, besides occurrence net equivalence, none of the equivalences mentioned above (including the partial order equivalences!) is real-time consistent. Therefore we introduce the notion of ST-bisimulation equivalence, which is real-time consistent. Moreover a complete proof system will be presented for those finite ST-bisimulation processes in which no action can occur concurrently with itself.
Note: Partial support received from the European Communities under ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (METEOR).

INTRODUCTION

One of the most controversial issues in the theory of concurrency is the issue of interleaving semantics versus True concurrency. People advocating interleaving semantics model the parallel composition of two processes by interleaving the atomic actions performed by these processes. A typical equation valid in interleaving semantics is: $a\|b = a\cdot b + b\cdot a$. The intended meaning of this equation is that when you observe the parallel composition of atomic actions a and b, either you see first the a and then the b, or you see first the b and then the a. The interleaving fans are aware of the fact that for some applications their approach is not realistic (for example in those places where real-time or fairness aspects are important). But since reality is extremely complicated one needs some simplifying assumptions, and they argue that the interleaving assumption is a good one: it allows for the construction of very elegant mathematical models of concurrency in which specification and verification of large concurrent systems is feasible.

The True concurrency adherents on the contrary think that interleaving models are very unrealistic. With an enormous enthusiasm they present all kinds of examples, formulated in terms of Petri nets, event structures or Mazurkiewicz traces, which show that the interleaving approach is bizarre.

Another controversy in the theory of concurrency which we would like to mention here is the issue of linear time versus branching time. It has to do with the equation: $a\cdot(b+c) = a\cdot b + a\cdot c$. Here the intended meaning is that when you observe first *a and then b* or *c*, this is just the same as when you observe a and then *b or a* and then c. Most True concurrency theories use this equation because the people who developed these theories thought there was nothing against it. The major part of the interleaving adherents however, reject the equation because they think that the timing of the choices in a process is often essential. They argue that for the proper description of features like deadlock behaviour, divergence, fair abstraction and interrupt mechanisms, information concerning the timing of choices is crucially important. Because we also think that very often the timing of choices is relevant, we are interested in branching time models.

The discussion in this paper takes place in the setting of ACP, the Algebra of Communicating Processes, as described in [BK1-3]. Until now all models studied in the ACP framework were based on the interleaving assumption and it was very difficult to deal with real-time aspects of concurrent systems. However, real-time aspects are often important (or even vital) in the design of concurrent systems. The major reason which motivated our excursion into the domain of True concurrency, was that doing this might help us answer the following question:

Can the language of ACP be used to specify real-time concurrent systems?

When True concurrency adherents argue that the interleaving approach is not realistic, the 'interleavers' answer that their approach at least makes algebraic system verification possible. This leads to our second question:

Is it possible to perform algebraic system verifications in a non-interleaved semantics?

Petri nets, introduced in 1962 by Petri in his now famous thesis ([Pe]), is the best known framework in which both True concurrency and branching time can be modelled. For this reason we have decided to define our models in terms

of Petri nets. Furthermore there are a lot of interesting theoretical results concerning Petri nets, which we might use. Last but not least we have chosen Petri nets because they have nice graphical representations. Pictures are important, not only from a didactical point of view, but also to make the analysis of the models easier.

A lot of people have been trying to combine ideas from CCS and ACP-like theories with True concurrency. NIELSEN, PLOTKIN & WINSKEL [NPW] were (as far as we know) the first ones who looked for a branching time non-interleaved model of concurrency. In [W1] an event structure semantics is presented for CCS and related languages. In [GM] a semantics for CCS is presented based on the notion of occurrence nets. [Po] presents non-interleaved versions of various equivalences like observation equivalence and failure equivalence, in the setting of Petri nets.

However, in none of these papers much attention is paid to *algebraic laws* valid in the various models. Only in a recent paper by BOUDOL & CASTELLANI [BC] this issue is studied. But their paper only deals with the case in which there is no communication possible between the components of a system (communication will be dealt with in a paper by them which is now in preparation).

In the first section of this paper we present a short review of part of the theory of ACP. Section 2 contains the basic definitions about Petri nets which we need in the rest of the paper. Moreover the fundamental notion of *occurrence net equivalence* is defined. In section 3 we define the ACP-operators on Petri nets. Some of the definitions are adopted from other authors, whereas some others (notably the sequential composition) are new here.

In section 4 we present our first algebraic model, which is based on the occurrence net equivalence. We will argue that in this model there are not enough laws valid to make practical system verifications possible. Still the occurrence net equivalence is important theoretically: it gives the identifications which, from an observational point of view, should be made in any case. In section 4 we also give the semantical notion of *bisimulation equivalence,* which leads to an interleaved Petri net model of the axiom system ACP, which is isomorphic to the graph model of BAETEN, BERGSTRA & KLOP [BBK1]. Finally section 4 contains a non-interleaved model for a subset of ACP, based on the notion of *concurrent bisimulation equivalence.* This equivalence, which can be situated between occurrence net and bisimulation equivalence, is also described by NIELSEN & THIAGARAJAN [NT] and GOLTZ [G]. All the equivalences in section 4 respect branching time.

In section 5 we add the concept of causality. Causality is not respected by the concurrent (or ordinary) bisimulation equivalence. We present the *pomset equivalence,* which is derived from PRATT [Pr1-2], and closely related to the trace theory of MAZURKIEWICZ [M1-2], see also [AR]. The pomset equivalence does respect causality but violates branching time. This brings us to the *pomset bisimulation equivalence,* which is inspired by BOUDOL & CASTELLANI [BC]. The pomset bisimulation equivalence respects both causality and branching time. However, we will argue that it does not fully respect the combination of both concepts. We propose a refinement of this notion which is more satisfactory in this sense.

Section 6 is a digression into the domain of real-time semantics. We give a real-time interpretation of process algebra, using the notion of timed Petri nets as presented in, for example, CARLIER, CHRETIENNE & GIRAULT [CCG]. New in our approach is the notion of *real-time consistency.* An equivalence relation on Petri nets is real-time consistent if it does not identify nets with a different real-time behaviour. A model based on a real-time consistent equivalence makes it possible to reason about concurrent systems in a real-time consistent manner without having to deal with the full complexity of real-time. A major result of this paper is that, except for the occurrence net equivalence, all previously described equivalences (including the partial order equivalences!) are *not* real-time consistent.

Therefore we present in section 7 a new equivalence, which *is* real-time consistent: the *ST-bisimulation equivalence.* This equivalence leads to a model for the ACP operators. We give a complete axiomatisation for finite processes in which no action can occur concurrently with itself. The structure of the complete axiomatisation is remarkable: if one wants to prove that two terms represent the same process in ST-bisimulation semantics, one first has to translate these terms into other ACP-terms. Thereafter one has to prove the equivalence of the new terms using the axioms of ACP (including the interleaving axiom CM1!!). If and only if this succeeds, the original terms are equivalent in the non-interleaved ST-bisimulation semantics. In our view the results of section 7 show that in principle the language of ACP can be used to specify real-time concurrent systems and to perform algebraic system verifications.

Due to lack of space almost all the proofs have been omitted. Also the topic of solving recursion equations is skipped here. For these issues we refer to the full paper [GV].

§1 A LANGUAGE FOR COMMUNICATING PROCESSES

In this section we present a language for reasoning about processes in an algebraical way - that is without referring to particular models - which is based on the axiom system ACP of BERGSTRA & KLOP [BK 1-3]. ACP starts from a collection A of given atomic actions. These actions are taken to be indivisible, and form a parameter of the axiom system. For each atomic action $a \in A$ there is a constant a in the language, representing the process, starting with an a-step and terminating after some time. Furthermore we have a special constant δ, denoting deadlock, the acknowledgement of a process that it cannot do anything any more. me absence of an alternative. We will write $A_\delta = A \cup \{\delta\}$. In ACP a process can end in two ways: by terminating successfully or by reaching a state of deadlock. Now processes can be built up from smaller ones by means of sequential, alternative and parallel composition operators. If x and y are two processes, then $x \cdot y$ is the process that starts the execution of y after completion of x (x must have terminated successfully), and $x + y$ is the process that can do either x or y. We do not specify whether the choice between the alternatives is made by the process itself, or by the environment, but it should be made at the beginning of $x + y$. The merge of

two processes, $x\|y$, executes the processes x and y simultaneously and independently, except for communication actions. In ACP, communication between parallel processes is modelled by means of a binary communication function $\gamma: A_\delta \times A_\delta \to A_\delta$, which is commutative, associative and has δ as zero element. γ forms a second (and last) parameter of the system. If $\gamma(a,b)=c\neq\delta$ this means that if a occurs in a process x and b occurs in y then in the merge $x\|y$ the processes x and y can communicate by synchronising the actions a and b. The synchronous performance of a and b is then regarded as a performance of the communication action c. In a merge, communication is never forced: the process $x\|y$ can either execute the actions a and b independently or perform the communication action c instead. This leaves open the possibility that x finds a communication partner in the environment, outside y, for instance in a context $(x\|y)\|z$. However, synchronisation can be forced by means of the encapsulation operator ∂_H. Here H is a set of atomic actions. Operator ∂_H blocks actions from H by means of a renaming into δ. It is used to *encapsulate* a process, i.e. to make communications with the environment impossible. In ACP there are also two auxiliary operators $\lfloor\!\lfloor$ and $|$, which are used for reducing concurrency to nondeterminism. Here we will skip them for the moment. Below we give the formal signature of our language, as well as some basic axioms.

1.1. Signature.

S (Sort):	P	(the set of processes)
F (Functions):	$+: P\times P\to P$	(alternative composition or sum)
	$\cdot: P\times P\to P$	(sequential composition or product)
	$\|: P\times P\to P$	(parallel composition or merge)
	$\partial_H: P\to P$	(encapsulation; $H\subseteq A$)
C (Constants):	$\delta\in P$	(deadlock)
	$a\in P$	(atomic actions; $a\in A$)

1.2. Note. In a product $x\cdot y$ we will often omit the $\cdot$. About leaving out parentheses: we take $\cdot$ to be more binding than other operations and $+$ to be less binding than other operations. In case of an associative operator, we also leave out parentheses.

1.3. Axioms. These are presented in table 1. Here $a\in A_\delta; x,y,z\in P; H\subseteq A$.

$x+y=y+x$	A1	$\partial_H(a)=a$ if $a\notin H$	D1
$x+(y+z)=(x+y)+z$	A2	$\partial_H(a)=\delta$ if $a\in H$	D2
$x+x=x$	A3	$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$	D3
$(x+y)z=xz+yz$	A4	$\partial_H(xy)=\partial_H(x)\cdot\partial_H(y)$	D4
$(xy)z=x(yz)$	A5		
$x+\delta=x$	A6	$x\|y=y\|x$	C1
$\delta x=\delta$	A7	$(x\|y)\|z=x\|(y\|z)$	C2

TABLE 1

1.4. Recursion. A *Recursive specification* E is a set of equations $\{x=t_x \mid x\in V_E\}$ with V_E a set of variables and t_x a process expression for $x\in V_E$. Only the variables of V_E may appear in t_x. A solution of E is an interpretation of the variables of V_E as processes (in a certain domain), such that the equations of E are satisfied.

The Recursive Definition Principle (RDP) tells us that every recursive specification has a solution. We conjecture that all models presented here satisfy RDP. For a substantiation of this conjecture we refer to [GV].

Recursive specifications are used to define (or specify) processes. Our language can be made recursive, by adding the syntactic constructs $\langle x\mid E\rangle$ (with $x\in V_E$), denoting the x-component of one of the solutions of E. This limits the class of models of the language to the ones satisfying RDP. However, in this extended abstract recursive constructs will not be used.

§2 PETRI NETS

In this section we will define the elements of the models that will be constructed in this paper. This is not the place to give an extensive introduction into the theory of Petri Nets. For this we refer to [R] or [RT].

2.1. DEFINITION: Let A be a given alphabet.

i) A *labelled marked net (over A)* is a 5-tuple $N=(S,T,F,M_{in},l)$ where:
- S is a set of *places* or *S-elements;*
- T is a set of *transitions* or *T-elements*, $S\cap T=\emptyset$;
- $F\subseteq S\times T\cup T\times S$, F is called the *flow relation*, $T\subseteq ran(F)$;
- $M_{in}: S\to\mathbb{N}$, M_{in} is called the *initial marking;*
- $l: T\to A$, l is called the *labelling function.*

ii) For $x\in S\cup T$ ${}^\bullet x=\{y\mid yFx\}$ is called the *preset* of x; $x^\bullet=\{y\mid xFy\}$ is called the *postset* of x.

2.2. The well known graphical representation of nets is as follows. Places and transitions are represented as circles and boxes, respectively. The flow relation is indicated by arcs between the corresponding circles and boxes. Transitions are inscribed by their labels. The initial marking is represented by placing dots (*tokens*) in the corresponding circles.

2.3. DEFINITION: Let $N=(S,T,F,M_{in},l)$ be a labelled marked net.

i) A function $M: S \to \mathbb{N}$ is called a *marking* of N;

ii) Let M be a marking. We say that $t \in T$ is *M-enabled* (*enabled*, to *occur* or to *fire*), notation $M[t>$, if $\forall s \in {}^\bullet t : M(s) \geq 1$;

iii) Let M, M' be markings and let $t \in T$. We say that M' *t-follows* M, and *results from firing t*, notation $M[t>M'$, if:
   a. $M[t>$
   b. $\forall s \in S$:

$$M'(s) = \begin{cases} M(s)-1 & \text{if } s \in {}^\bullet t - t^\bullet \\ M(s)+1 & \text{if } s \in t^\bullet - {}^\bullet t \\ M(s) & \text{otherwise} \end{cases}$$

iv) Let M be a marking. A sequence $\alpha = t_1 \star \cdots \star t_n \in T^*$ is *M-enabled*, notation $M[\alpha>$, if there exist $M_0, \ldots, M_n$ such that $M_0 = M$ and for $1 \leq i \leq n$: $M_{i-1}[t_i > M_i$. We say that M_n is *obtained from M by firing* α, notation $M[\alpha > M_n$. We also say that M_n is *reachable* from M;

v) Let M, M' be markings, let $\alpha = t_1 \star \cdots \star t_n \in T^*$, and let $M[\alpha > M'$. In this case we say that the sequence $\sigma = l(t_1) \star \cdots \star l(t_n) \in A^*$ is *M-enabled*, notation $M[\sigma>$. We also say that M' *is obtained from M by firing* σ, notation $M[\sigma > M'$;

vi) For each marking M, $[M>$ is the set of markings reachable from M;

vii) N is called *safe* if for each $M \in [M_{in}>$ and for each $s \in S$: $M(s) \leq 1$. In this case each marking reachable from M_{in} can be considered as a subset of S.

2.4. DEFINITION: Let $N=(S,T,F,M_{in},l)$ be a safe labelled marked net.

i) Let M be a reachable marking of N and let U be a subset of T. We say that U is *M-enabled*, notation $M[U>$, if:

   a. $\forall t \in U \; \forall s \in {}^\bullet t : M(s) = 1$

   b. $\forall t,t' \in U : t \neq t' \Rightarrow {}^\bullet t \cap {}^\bullet t' = \varnothing$ (Because N is safe, this implies ${}^\bullet t \cap (t')^\bullet = \varnothing$.)

ii) N has *bounded parallelism* if for every $M \in [M_{in}>$, and $U \subseteq T$ such that $M[U>$, we have that U is finite.

2.5. Note. In this paper we will only consider safe labelled marked nets, which have bounded parallelism and a non-empty initial marking. We will use the word *net* to denote such a structure. For $i=0,1,...,$ and nets N_i, we will denote the components of N_i by resp. S_i, T_i, F_i, $(M_{in})_i$ and l_i. Let λ be some infinite cardinal, and let A be a given set of atomic actions. $\mathbb{N}_\lambda(A)$ is the set of nets over alphabet A such that the cardinality of the sets of places and transitions is less than λ.

2.6. Occurrence nets. Above we have introduced the notion of a Petri net. Furthermore we have defined the dynamic behaviour of a net in terms of its markings and the firing rule. Most Petri nets have the property that the state (marking) which results after a number of transitions has been fired, gives us almost no information about this firing sequence. Consider for example the net in figure 1:

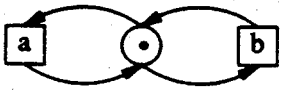


FIGURE 1

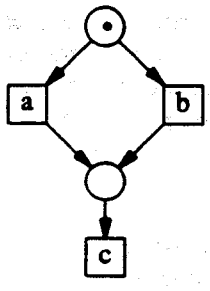


FIGURE 2

There is only one state, so we have absolutely no information about the firing sequence of *a*'s and *b*'s leading to it. Figure 2 presents another example. The state which results from firing the *a*-transition, could have been caused also by the firing of the *b*-transition.

Below we will define the fundamental notion of occurrence nets - a subclass of acyclic nets with places which can only be forwardly branched. Occurrence nets have the property that each transition can fire only once. Furthermore we can deduce from the state of the net which were the transitions leading to it. Occurrence nets were introduced in NIELSEN, PLOTKIN & WINSKEL [NPW]. Our definition is taken from WINSKEL [W2].

2.6.1. DEFINITION: An *occurrence net* is a net (S,T,F,M_{in},l) for which the following restrictions are satisfied:

1) $\forall s\in S: |{}^\bullet s|\leq 1$;

2) $s\in M_{in} \Leftrightarrow {}^\bullet s=\emptyset$;

3) F^+ (the transitive closure of F) is irreflexive and $\forall t\in T:\{t'\in T \mid t'F^\bullet t\}$ is finite;

4) The *conflict relation* $\sharp$ is irreflexive, where for $x,y\in S\cup T$: $x\sharp y \Leftrightarrow \exists t,t'\in T: t\neq t'$, ${}^\bullet t\cap {}^\bullet t'\neq\emptyset$, $tF^\bullet x$ and $t'F^\bullet y$.

2.6.2. PROPOSITION: *Let $N=(S,T,F,M_{in},l)$ be an occurrence net. Then: $\forall t\in T\ \exists M\in[M_{in}>: M[t>$ and $\forall s\in S\ \exists M\in[M_{in}>: s\in M$. In words: N does not contain superfluous transitions or places.*

2.7. Unfolding. By means of the notion of *unfolding* we will now relate to each net an occurrence net. As pointed out by [NPW] and [W2], the behaviour of the unfolding of a net is exactly the same as the behaviour of the net itself; the only difference is that the unfolding has a "memory" in which the dynamic history is stored. Definition 2.7.1 is rather technical, but the reader can learn the essential properties of the unfolding operator from examples 2.7.2 and theorem 2.8.

2.7.1. DEFINITION: Let N_0 be a net. The *unfolding* of N_0, notation $\mathfrak{U}(N_0)$, is the net N_1 defined inductively as follows. $S_1,T_1,F_1,(M_{in})_1$ and $\mathfrak{R}$ are the least sets satisfying:

1. $(M_{in})_1\subseteq S_1\subseteq\{(t,s)\mid t\in T_1\cup\{*\},s\in S_0\}$;
2. $T_1\subseteq\{(M,t)\mid M\subseteq S_1,t\in T_0\}$;
3. $F_1\subseteq S_1\times T_1\cup T_1\times S_1$;
4. $\mathfrak{R}\subseteq Pow(S_1)$;
5. $(M_{in})_1=\{(*,s)\mid s\in(M_{in})_0\}\in\mathfrak{R}$;
6. If, for some index set I, $\{(t_i,s_i)\mid i\in I\}\in\mathfrak{R}$ and $\{s_i\mid i\in I\}[t>$ then:

$$t'=(\{(t_i,s_i)\mid i\in I \text{ and } s_i\in{}^\bullet t\},t)\in T_1$$

$$\{(t',s)\mid s\in t^\bullet\}\subseteq S_1$$

$$\{(t_i,s_i)\mid i\in I \text{ and } s_i\notin{}^\bullet t\}\cup\{(t',s)\mid s\in t^\bullet\}\in\mathfrak{R}$$

$$\{((t_i,s_i),t')\mid i\in I \text{ and } s_i\in{}^\bullet t\}\cup\{(t',(t',s))\mid s\in t^\bullet\}\subseteq F_1$$

l_1 is the function from T_1 into A defined by: $(m,t)\in T_1 \Rightarrow l_1((m,t))=l_0(t)$.

2.7.2. EXAMPLES:

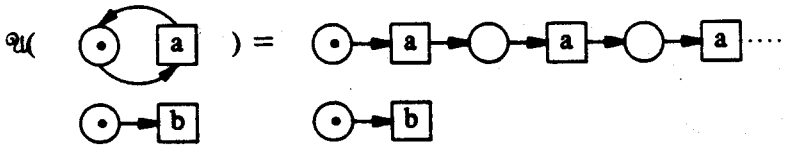


FIGURE 3

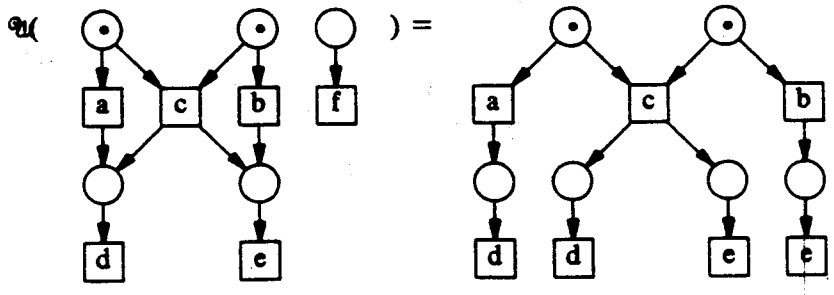


FIGURE 4

2.8. THEOREM: *Let N_0 be a net, let $N_1=\mathfrak{U}(N_0)$, and let $\mathfrak{R}$ be as defined in section 2.7.1. Then:*

i) $\mathfrak{R}=[(M_{in})_1>$;

ii) *N_1 is an occurrence net;*

iii) *If N_0 is an occurrence net, then N_1 is isomorphic to N_0 (two nets are considered isomorphic if they only differ in the names of places and transitions).*

2.8.1. REMARK: For $\lambda>\aleph_0$ we have that $\mathfrak{U}: \mathbf{N}_\lambda(A)\rightarrow\mathbf{N}_\lambda(A)$.

2.9. DEFINITION: Two nets N_0 and N_1 are *occurrence net equivalent*, notation $N_0 \equiv_{occ} N_1$, iff $\mathfrak{U}(N_0)$ is isomorphic to $\mathfrak{U}(N_1)$.

Clearly, $\equiv_{occ}$ is an equivalence relation. Note that theorem 2.8 implies that in each equivalence class of $\equiv_{occ}$ there is, up to isomorphism, exactly one occurrence net.

2.10. Root-unwinding. For the proper definition of some of the process operators we need to work with nets which have acyclic roots, i.e. the places of the initial marking have no incoming arrows. What we could do in order to reach this situation is a restriction of the domain to a subset of nets which have acyclic roots, for example to occurrence nets. Or we could start the definitions of the operations with the unfolding of the nets involved. We do not like the first solution because we think that in a lot of applications it is very natural to work with nets with cyclic roots. The second solution is also not favoured by us since the operation of unfolding a net is very complicated. This means that, in order to add two nets one has to perform an awful amount of work. Moreover, the operation of unfolding refers heavily to the token-game and is therefore very operational. We are interested in a *general* non-operational construction that relates, with a minimum amount of work, to every net an occurrent equivalent net with acyclic roots. The simplest solution we could find is the *root-unwinding* operation presented below. The construction resembles one in GOLTZ [G], but, due to the fact that our nets can be infinite and we want to stay in axiomatic set theory, our construction is a bit more complicated. But as explained above, if someone does not like the operation he or she can just skip the definition and work with unfoldings instead.

2.10.1. DEFINITION: We define the *root-unwinding* map $\mathfrak{R}: \mathbf{N}_\lambda(A) \to \mathbf{N}_\lambda(A)$ as follows. Let $N_0 \in \mathbf{N}_\lambda(A)$. Let $M_{cyc} = \{s \in (M_{in})_0 \mid {}^\bullet s \neq \emptyset\}$ be the set of cyclic root elements, and let $M^c_{cyc} = \{s^c \mid s \in M_{cyc}\}$ be a copy of this set. Then $\mathfrak{R}(N_0)$ is the net N_1 obtained by adding the places in M^c_{cyc}, and put them in the initial marking instead of the elements of M_{cyc}:

$$S_1 = S_0 \cup M^c_{cyc} \quad \text{and} \quad (M_{in})_1 = ((M_{in})_0 - M_{cyc}) \cup M^c_{cyc}$$

We define a set $\mathbf{U}$ by:

$$\mathbf{U} = \{M_{cyc} - (\bigcup_{t \in X} {}^\bullet t) \mid X \text{ a finite subset of } T_0\}$$

Note that the cardinality of $\mathbf{U}$ is less than λ. For each $U \in \mathbf{U}$ and $t \in T_0$ such that ${}^\bullet t \cap U$ is non-empty, we introduce a new transition $<U,t>$ such that:

$${}^\bullet <U,t> = ({}^\bullet t - U) \cup \{s^c \mid s \in {}^\bullet t \cap U\} \quad \text{and} \quad <U,t>^\bullet = t^\bullet$$

The label of $<U,t>$ is the label of t. Because the cardinality of $\mathbf{U}$ is less than λ, the number of new transitions in the root-unwinding N_1 is also less than λ. Thus we avoid the cardinality problem that arises if we introduce a new transition $<U,t>$ for every subset U of M_{cyc}.

2.10.2. EXAMPLE:

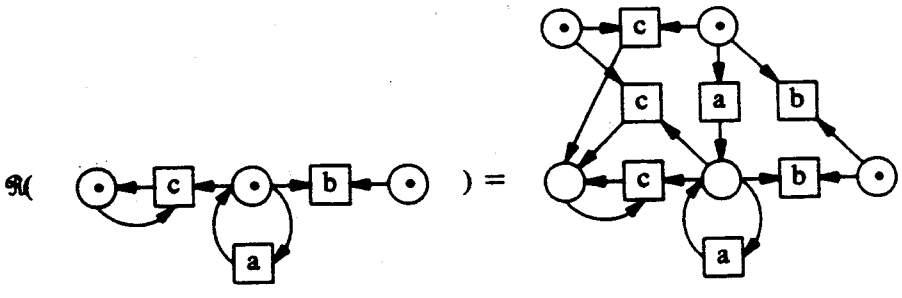


FIGURE 5

2.10.3. LEMMA: *Let $N_0 \in \mathbf{N}_\lambda(A)$ and let $N_1 = \mathfrak{R}(N_0)$. Then:*

i) $N_0 \equiv_{occ} N_1$;

ii) $\forall s \in (M_{in})_1 : {}^\bullet s = \emptyset$;

iii) *If* $\forall s \in (M_{in})_0 : {}^\bullet s = \emptyset$, *then* N_1 *is identical to* N_0.

§3 DEFINITION OF ACP OPERATORS ON NETS

In this section we will define the operators of section 1 on the set $\mathbb{N}_\lambda(A)$ of nets. All the models which are presented in this paper are constructed by dividing out an equivalence relation on nets, which is a congruence with respect to the operators as defined here.

3.1. An atomic action $a \in A$ is interpreted as:

3.2. δ is represented by the net:

Execution of the process associated with a net terminates when, after firing a number of transitions, no token is left. So the process a terminates after the firing of its transition. The process δ however cannot terminate because there is no transition that can remove the token of the initial marking.

3.3.1. $+$: The definition of the $+$ is from GOLTZ & MYCROFT [GM]. By means of a cartesian product construction, conflicts are introduced between all pairs of initial transitions of the two nets involved. Let $N_0, N_1 \in \mathbb{N}_\lambda(A)$, $N_2 = \mathfrak{R}(N_0)$ and $N_3 = \mathfrak{R}(N_1)$. $N_4 = N_0 + N_1$ is defined by:

$$S_4 = (S_2 - (M_{in})_2) \dot\cup (S_3 - (M_{in})_3) \dot\cup (M_{in})_2 \times (M_{in})_3$$

$$T_4 = T_2 \dot\cup T_3$$

$$F_4 = ((F_2 \dot\cup F_3) \cap (S_4 \times T_4 \cup T_4 \times S_4)) \dot\cup \{((s_0,s_1),t) \mid (s_0,t) \in F_2 \text{ or } (s_1,t) \in F_3\}$$

$$(M_{in})_4 = (M_{in})_2 \times (M_{in})_3$$

$$l_4 = l_2 \dot\cup l_3$$

3.3.2. EXAMPLE:

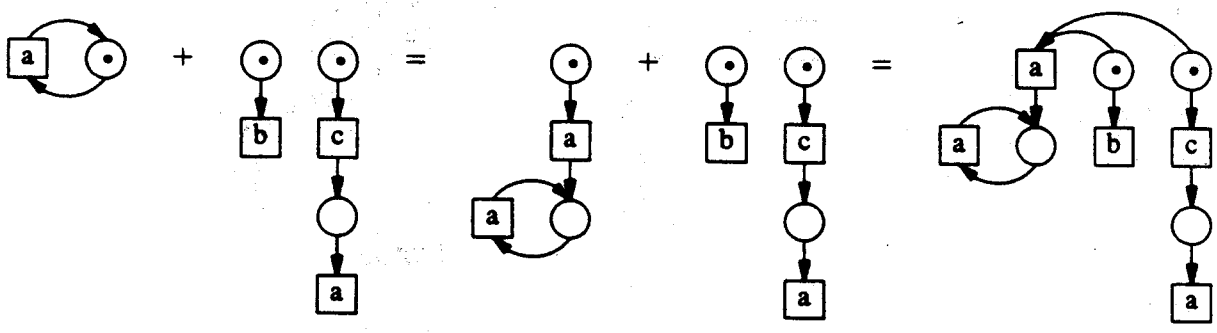


FIGURE 6

3.4.1. $\cdot$: Below we present a definition of the $\cdot$-operator. In the definition we use the well-known construction of *complements*: for each place we introduce a number of complementary places in such a way that a place contains a token iff the complementary places are empty. If all the new places contain a token we know that the old places are empty, and thus execution of the first process is finished. If $N_0, N_1 \in \mathbb{N}_\lambda(A)$ and $N_2 = \mathfrak{R}(N_1)$, define $N_3 = N_0 \cdot N_1$ by:

$$S_3 = S_0 \dot\cup (S_2 - (M_{in})_2) \dot\cup (S_0 \times (M_{in})_2)$$

$$T_3 = T_0 \dot\cup T_2$$

$$(M_{in})_3 = (M_{in})_0 \dot\cup (S_0 - (M_{in})_0) \times (M_{in})_2$$

Besides the arrows of F_0 and $F_2 \cap (S_3 \times T_3 \cup T_3 \times S_3)$, F_3 contains arrows defined by:

$$[(s,t) \in F_0 \wedge (t,s) \notin F_0 \wedge s' \in (M_{in})_2] \Rightarrow (t,(s,s')) \in F_3$$

$$[(t,s) \in F_0 \wedge (s,t) \notin F_0 \wedge s' \in (M_{in})_2] \Rightarrow ((s,s'),t) \in F_3$$

$$[(s_0,s_1) \in S_0 \times (M_{in})_2 \wedge (s_1,t) \in T_2] \Rightarrow ((s_0,s_1),t) \in F_3$$

The labelling function is given by: $l_3 = l_0 \dot\cup l_2$.

3.4.2. EXAMPLE:

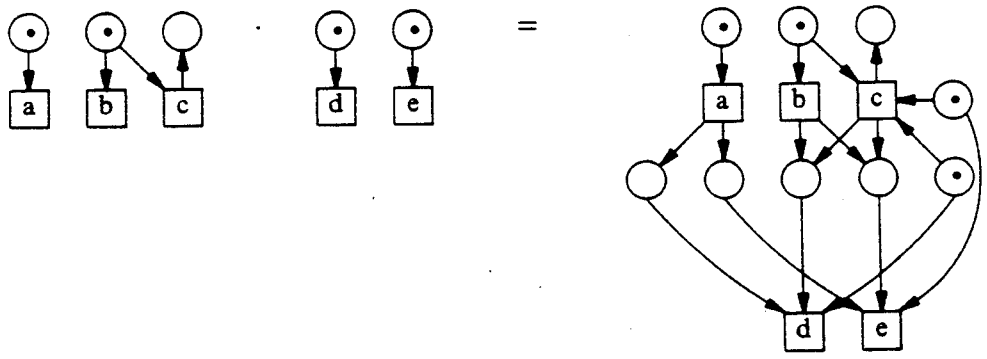


FIGURE 7

3.5. ||: If $N_0, N_1 \in \mathbf{N}_\lambda(A)$, obtain $N_2 = N_0 \| N_1$ by taking the disjoint union of N_0 and N_1, adding transitions:

$$\{(t,u) \mid t \in T_0,\ u \in T_1,\ \gamma(l_0(t), l_1(u)) \neq \delta\}$$

and taking: $l_2((t,u)) = \gamma(l_0(t), l_1(u))$, ${}^\bullet(t,u) = {}^\bullet t \cup {}^\bullet u$ and $(t,u)^\bullet = t^\bullet \cup u^\bullet$.

3.6.1. ∂_H: Let $N \in \mathbf{N}_\lambda(A)$. $\partial_H(N)$ is obtained from N by omitting the transitions with labels in H and the flow pairs they occur in.

3.6.2. EXAMPLE: Let $\gamma(a,b) = c$, and let $H = \{a,b\}$. Then:

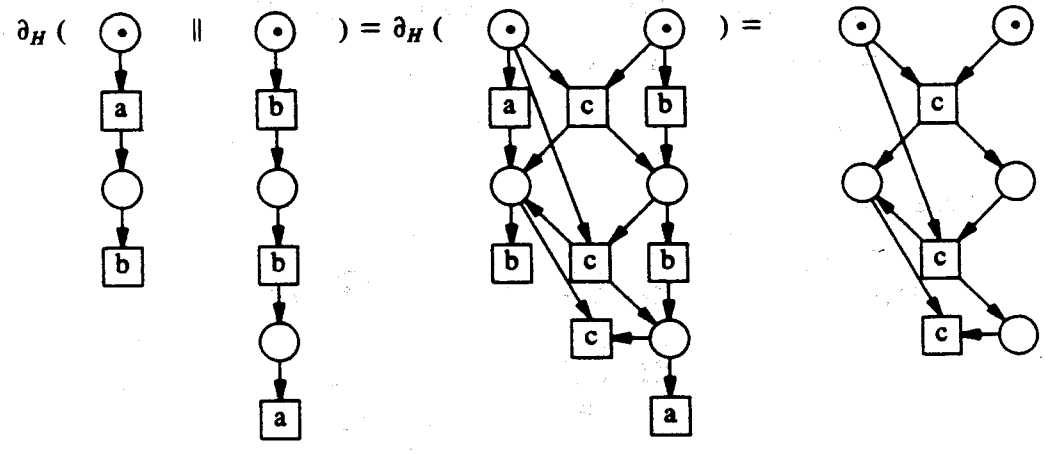


FIGURE 8

§4 OCCURRENCE NET, BISIMULATION AND CONCURRENT BISIMULATION SEMANTICS

In section 3 we defined the operators of section 1 on the Petri net domains of section 2. Thus we are able to associate nets to process expressions in a compositional way. Now a Petri net model can be obtained by dividing out a congruence relation on a domain $\mathbf{N}_\lambda(A)$. In such a model several processes - which are regarded to have the same behaviour - are identified. In fact the suitability of a congruence is determined by those aspects of the behaviour of concurrent systems we are interested in. Since we want to build proof systems for verifying that two processes have the same behaviour, and since we will use the constructed models as a criterion for establishing the validity of the proof systems, it is no good solution not to identify any processes at all (to select the identity as congruence). On the contrary, our purpose is to identify any two processes for which we can find no interesting property that discriminates between them. Of course it is application dependent which properties should be considered as interesting. Therefore we will describe a variety of possibilities. Each of the equivalences mentioned in this paper gives rise to another process semantics.

4.1. Occurrence net semantics.

The least identifying semantics of this paper is based upon occurrence net equivalence, as defined in section 2.9. Occurrence net semantics was already employed by NIELSEN, PLOTKIN & WINSKEL [NPW], WINSKEL [W2] and GOLTZ & MYCROFT [GM]. It carefully respects the local structure of nets, but the identity of places and transitions is not preserved. However, in an algebraic approach in which we have operators like + and ·, this loss of identity is unavoidable, because the definition of these operators only makes sense if we identify a net N with $\mathfrak{R}(N)$.

4.1.1. PROPOSITION: *$\equiv_{occ}$ is a congruence w.r.t. the operators of section 1.*

4.1.2. PROPOSITION: *$\mathbf{N}_\lambda(A) / \equiv_{occ}$ satisfies the axioms of table 1, except for axioms A3, A4 and A5.*

As illustrated in figure 9, the model $\mathbf{N}_\lambda(A)/\equiv_{\ldots}$ does not even identify processes like a and $a+a$, which behave very much the same. Hence we argue that in this model there are not enough laws valid to make practical system verifications possible.

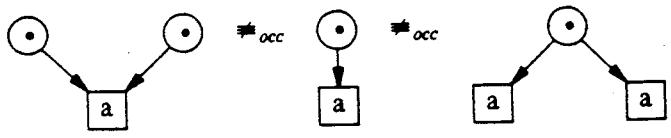


FIGURE 9

4.2. Bisimulation semantics.

In the occurrence net semantics of the previous section the structure of processes is carefully respected, but, due to the low degree of identification, there are not enough laws for algebraic system verification. Now we will look at an opposite point in our spectrum. *Bisimulation semantics* is firmly based upon the interleaving assumption. Consequently, some features of concurrency, like causal dependencies, cannot be modeled any more. However, for this semantics rich algebraic theories are available which can be - and have been - used for verification of several kinds of systems.

As far as this paper is concerned, the virtue of bisimulation semantics is twofold. Firstly, it gives rise to powerful algebraic tools which can be used for all applications in which properties like causality and real-time behaviour are unimportant and the interleaving assumption is harmless. And secondly, it will turn out to be a valuable expedient for axiomatising non-interleaved semantics.

We start by defining *bisimulation equivalence* or *bisimilarity*. The original notion, although defined on graphs instead of nets, is due to PARK [Pa]. It can be regarded as a refinement of MILNER's *observational equivalence* [Mi]. Clause 4 below represent an idea from the ACP framework: the distinction between deadlock and successful termination (cf. section 1.0, section 3.2, and figure 12).

4.2.1. DEFINITION: Let $N_0, N_1 \in \mathbf{N}_\lambda(A)$. A relation $R \subseteq Pow(S_0) \times Pow(S_1)$ is a *bisimulation* between N_0 and N_1, notation $R: N_0 \leftrightarrows N_1$, if:

1. $(M_{in})_0 \; R \; (M_{in})_1$;
2. if $M_0 \; R \; M_1$ and $M_0[t_0 > M'_0$, then there are $t_1 \in T_1$ and $M'_1 \subseteq S_1$ such that $l_0(t_0) = l_1(t_1)$, $M_1[t_1 > M'_1$ and $M'_0 \; R \; M'_1$;
3. as 2 but with the roles of N_0 and N_1 reversed;
4. $M_0 \; R \; M_1 \;\Rightarrow\; (M_0 = \emptyset \Leftrightarrow M_1 = \emptyset)$.

N_0 and N_1 are *bisimilar*, notation $N_0 \leftrightarrows N_1$, iff there is an $R: N_0 \leftrightarrows N_1$.

4.2.2. EXAMPLES:

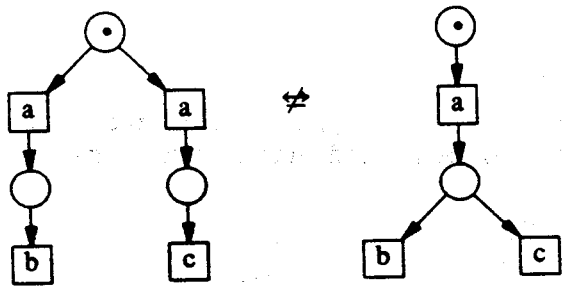


FIGURE 10

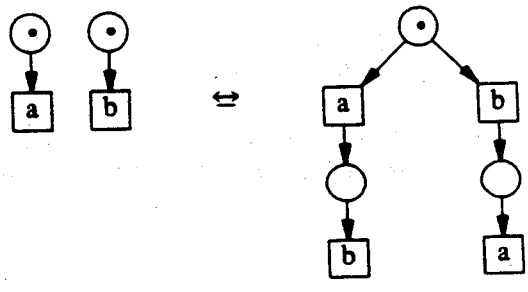


FIGURE 11

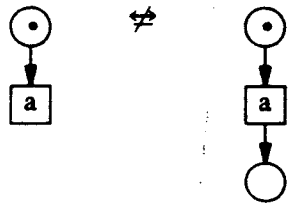


FIGURE 12

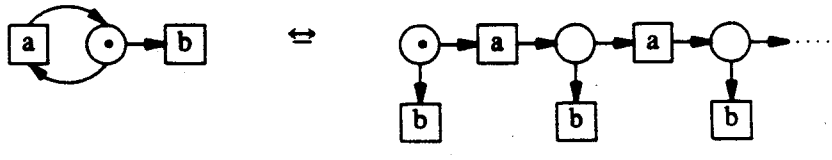


FIGURE 13

4.2.3. PROPOSITION: *$\leftrightarrows$ is an equivalence relation on $\mathbf{N}_\lambda(A)$, which is a congruence w.r.t. the operators of section 1.*

4.2.4. PROPOSITION: *$\mathbf{N}_\lambda(A)/\leftrightarrows$ satisfies the axioms of table 1.*

4.2.5. PROPOSITION: *Let $N \in \mathbf{N}_\lambda(A)$. Then $N \leftrightarrows \mathcal{U}(N)$. So $\equiv_{occ} \;\Rightarrow\; \leftrightarrows$.*

4.3. Petri nets and process graphs.
Since bisimulation semantics is based on the interleaving assumption, the model presented above does not really use the expressive power of Petri nets. In fact we claim that for $\lambda > \aleph_0$, the model $\mathbf{N}_\lambda(A)/\leftrightarrows$ is isomorphic to the graph model $\mathbf{G}_\lambda(A)/\leftrightarrows$ of [BBK1] (if we omit all τ's from this model). The usefulness of an interleaved Petri net model is (in our view) establishing a connection between the 'classical' graph models and the non-interleaved Petri net models in this paper. Furthermore, we think that the definition of the parallel composition operator is more natural in the Petri net model than it is in the graph model. Below, we try to make the relation between the two models more explicit.

4.3.1. DEFINITION: Let $N_0 \in \mathbf{N}_\lambda(A)$. The *sequentialisation* of N_0, notation $\mathbb{S}(N_0)$, is the net N_1 defined by:

$$S_1 = [(M_{in})_0> - \{\emptyset\}$$
$$T_1 = \{(M,t) \mid M \in S_1,\ t \in T_0 \text{ and } M[t>\}$$
$$F_1 = \{(M,(M,t)) \mid (M,t) \in T_1\} \cup \{((M,t),M') \mid (M,t) \in T_1,\ M[t>M' \text{ and } M' \neq \emptyset\}\ .$$
$$(M_{in})_1 = \{(M_{in})_0\}$$
$$l_1((M,t)) = l_0(t) \text{ for } (M,t) \in T_1$$

Because each reachable marking M of net N_0 can be characterized by a sequence $\sigma \in T^*$ with $(M_{in})_0[\sigma>M$, we have that the cardinality of S_1 is less than λ. Thus $\mathbb{S}$ is a function from $\mathbf{N}_\lambda(A)$ into $\mathbf{N}_\lambda(A)$.

4.3.2. EXAMPLE:

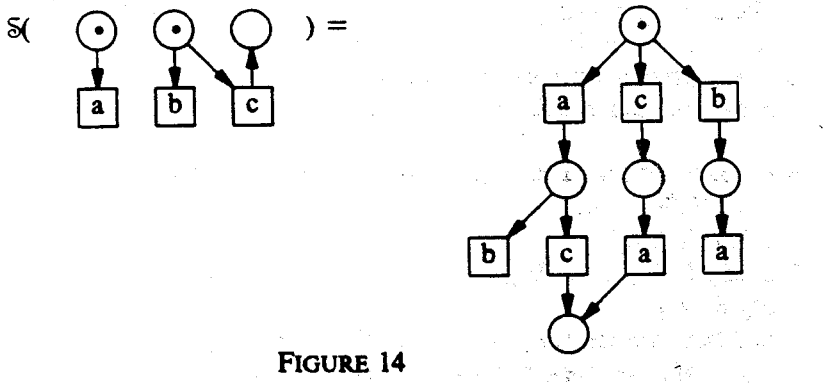


FIGURE 14

4.3.3. LEMMA: *Let $N_0 \in \mathbf{N}_\lambda(A)$ and let $N_1 = \mathbb{S}(N_0)$. Then:*
i) $N_0 \leftrightarrows N_1$;
ii) $|(M_{in})_1| = 1$;
iii) $\forall t \in T_1 : |{}^\bullet t| = 1$ *and* $|t^\bullet| \leq 1$;
iv) $\mathbb{S}(N_1)$ *is isomorphic to* N_1.

4.3.4. REMARK: Lemma 4.3.3 says that each element of $\mathbf{N}_\lambda(A)/\leftrightarrows$ contains a process-graph-like Petri net. In fact there is a 1-1 correspondence between sequentialisations of nets and τ-less process graphs. This is illustrated below.

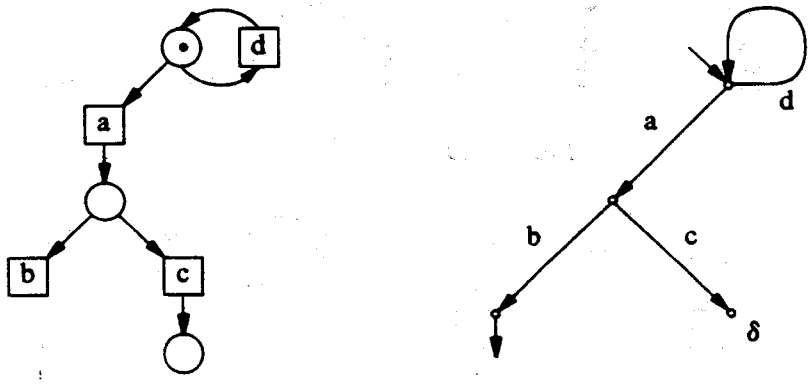


FIGURE 15

4.4. A complete proof system.
Now we will present a sound and complete proof system for bisimulation semantics; that is, a set of rules and axioms such that an equation $t = t'$ holds in the model $\mathbf{N}_\lambda(A)/\leftrightarrows$ if and only if it can be proved from the rules and axioms in this set. The proof system consists of the equational theory ACP of BERGSTRA & KLOP [BK1-3] (which deals with closed terms) plus a limit rule for open terms. In ACP we use two auxiliary operators $\lfloor\!\lfloor$ (left-merge) and $|$ (communication merge). The process $x \lfloor\!\lfloor y$ is the process $x \| y$, but with the restriction that the first step comes from x, and $x | y$

is $x\|y$ with a communication step as the first step. These operators are used in the reduction of concurrency to non-determinism, which is the basic simplification introduced by the interleaving approach.

4.4.1.1. $\llcorner$: If $N_0,N_1 \in \mathbf{N}_\lambda(A)$, obtain $N_2 = N_0 \llcorner N_1$ by taking $N_0\|N_1$ and adding a new S-element *ROOT*, which will become the only element of the initial marking of N_2. For each $t \in T_0$ and $M \subseteq S_0$ for which $(M_{in})_0[t>M$, we add a new T-element t^0 and put:
- $l_2(t^0) = l_0(t)$
- ${}^\bullet(t^0) = \{ROOT\}$ and $(t^0)^\bullet = M \cup (M_{in})_1$

4.4.1.2. EXAMPLE: Let $\gamma(a,b)=c$. Then:

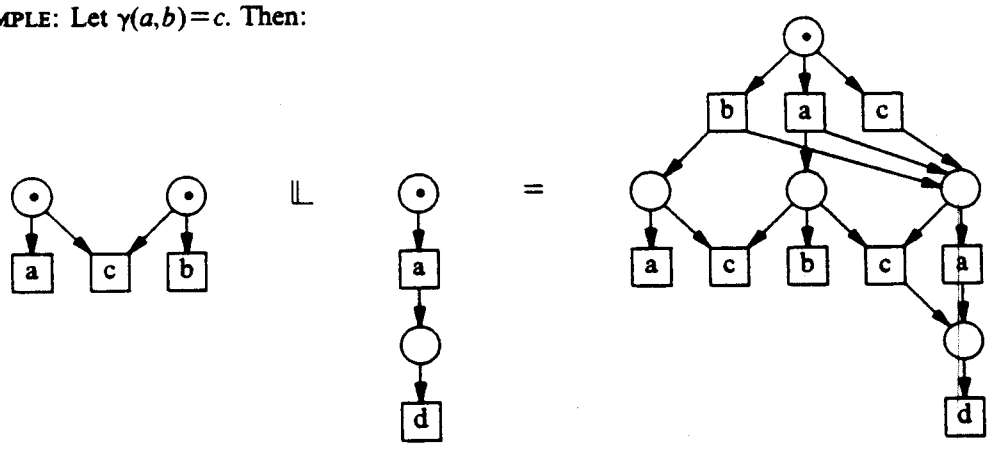


FIGURE 16

4.4.2.1. $|$: If $N_0,N_1 \in \mathbf{N}_\lambda(A)$, obtain $N_2 = N_0 | N_1$ by taking $N_0\|N_1$ and adding a new S-element *ROOT*, which will become the only element of the initial marking of N_2. Let $(M_{in})_0[t_0>M_0$, $(M_{in})_1[t_1>M_1$ and $\gamma(l_0(t_0),l_1(t_1))=c\neq\delta$. Then we add a new T-element (t_0,t_1) and put:
- $l_2((t_0,t_1)) = c$
- ${}^\bullet(t_0,t_1) = \{ROOT\}$ and $(t_0,t_1)^\bullet = M_0 \cup M_1$

4.4.2.2. EXAMPLE: Let $\gamma(a,b)=c$. Then:

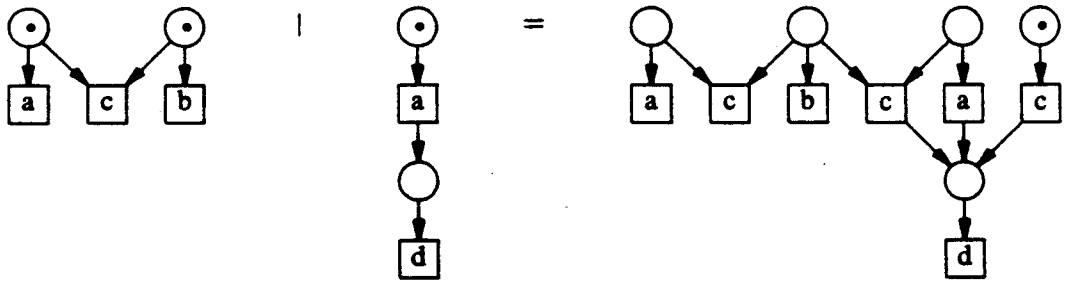


FIGURE 17

4.4.3. ACP consists of the axioms A and D of table 1 and the axioms CM below.

$x\|y = x\llcorner y + y\llcorner x + x\|y$	CM1
$a\llcorner x = ax$	CM2
$(ax)\llcorner y = a(x\|y)$	CM3
$(x+y)\llcorner z = x\llcorner z + y\llcorner z$	CM4
$(ax)\|b = (a\|b)x$	CM5
$a\|(bx) = (a\|b)x$	CM6
$(ax)\|(by) = (a\|b)(x\|y)$	CM7
$(x+y)\|z = x\|z + y\|z$	CM8
$x\|(y+z) = x\|y + x\|z$	CM9
$a\|b = \gamma(a,b)$	CM10

TABLE 2

Furthermore, we have the *limit rule* (LR) of BAETEN & BERGSTRA [BB]. It states that if an equation holds for all finite processes (i.e. if it holds after any substitution of closed terms for the variables in this equation) then it holds for all processes. Now the axioms C of table 1 are derivable from ACP+LR.

4.4.4. PROPOSITION: $\leftrightarrows$ *is also a congruence w.r.t. the operators* $\lfloor\!\lfloor$ *and* $|$.
4.4.5. PROPOSITION: $\mathbb{N}_\lambda(A)/\leftrightarrows \models \mathrm{ACP}+\mathrm{LR}$, *i.e.* $\mathbb{N}_\lambda(A)/\leftrightarrows$ *is a model of* ACP+LR.
4.4.6. THEOREM: $\mathbb{N}_\lambda(A)/\leftrightarrows \models t=t' \Leftrightarrow \mathrm{ACP}+\mathrm{LR} \vdash t=t'$, *i.e.* ACP+LR *is a sound and complete proof system.*

4.5. Concurrent bisimulation semantics.
In this section we present a non interleaved variant of bisimulation semantics. This *concurrent bisimulation semantics* takes explicitly into account the possibility that in a process like $a\|b$ the actions a and b occur concurrently (besides the possibilities that they occur one after the other, or synchronise into a communication action $\gamma(a,b)$). Definition 4.5.2 of the concurrent bisimulation, but without our termination clause 4, also appeared in [NT] and [G].

4.5.1. DEFINITION: Let $N = (S,T,F,M_{in},l)$ be a net, let M,M' be markings of N and let U be a subset of T. We say that M' *U-follows* M, and *results from firing the transitions in* U *concurrently,* notation $M[U>M'$, if:
a. $M[U>$ (see definition 2.4)
b. $\forall s\in S$:

$$M'(s) = \begin{cases} M(s)-1 & \text{if } \exists t\in U: s\in {}^\bullet t - t^\bullet \\ M(s)+1 & \text{if } \exists t\in U: s\in t^\bullet - {}^\bullet t \\ M(s) & \text{otherwise} \end{cases}$$

Note that if M is reachable and $M[U>M'$ then U must be finite (since N displays only bounded parallelism, see 2.5) so M' is reachable again.

4.5.2. DEFINITION: Let $N_0,N_1\in\mathbb{N}_\lambda(A)$. A relation $R\subseteq Pow(S_0)\times Pow(S_1)$ is a *concurrent bisimulation* between N_0 and N_1, notation $R:N_0\leftrightarrows_c N_1$, if:
1. $(M_{in})_0\ R\ (M_{in})_1$;
2. if $M_0\ R\ M_1$ and $M_0[U_0>M'_0$, then there are $U_1\subseteq T_1$ and $M'_1\subseteq S_1$ such that $l_0(U_0)=l_1(U_1)$, $M_1[U_1>M'_1$ and $M'_0\ R\ M'_1$ (here $l_i(U_i)$ denotes the multiset of labels of the transitions in U_i);
3. as 2 but with the roles of N_0 and N_1 reversed;
4. $M_0\ R\ M_1 \Rightarrow (M_0=\emptyset \Leftrightarrow M_1=\emptyset)$.

N_0 and N_1 are *concurrently bisimilar,* notation $N_0\leftrightarrows_c N_1$, iff there is an $R:N_0\leftrightarrows_c N_1$.

4.5.3. PROPOSITION: $\leftrightarrows_c$ *is an equivalence relation on* $\mathbb{N}_\lambda(A)$, *which is a congruence w.r.t. the operators of section 1.*
4.5.4. PROPOSITION: $\mathbb{N}_\lambda(A)/\leftrightarrows_c$ *satisfies the axioms of table 1.*
4.5.5. PROPOSITION: $\equiv_{occ} \Rightarrow \leftrightarrows_c \Rightarrow \leftrightarrows$.

4.5.6. EXAMPLE:

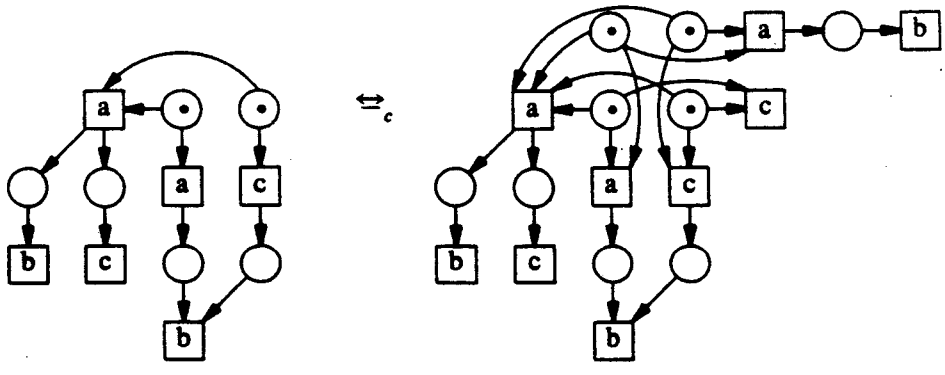


FIGURE 18

4.5.7. REMARK: The algebraic equivalent of figure 18 is the identity:

$$a(b\|c)+(a\|c)b = a(b\|c)+(a\|c)b+ab\|c$$

Here we see that the algebraic approach allows us to represent a rather complex Petri net by means of a relatively simple formula.

§5 PARTIAL ORDER SEMANTICS

The equation

$$a\|b = ab+ba, \tag{i}$$

which holds in the interleaved bisimulation semantics of section 4.2, is rejected in the concurrent bisimulation semantics of section 4.5. The reason for rejecting this equation is that the left hand side leaves open the possibility that a and b occur simultaneously, whereas this option is not present at the right hand side. The True concurrency adherents

also reject this equation, but for a more fundamental reason: in $a\|b$ the actions a and b occur *independently*, while in $ab+ba$ there is a *causal* link between them: a occurs before b or b occurs before a. In order to point out the difference more sharply the following notation will be used.

All models of concurrency investigated in this paper (except the one based on occurrence equivalence) satisfy the axioms A1-3 from table 1, expressing that the alternatives in a choice can be regarded to form a set. Now write $x \subseteq y$ for $x+y=y$. From A1, A2 and A3 respectively it follows that $\subseteq$ is antisymmetrical, transitive and reflexive, and hence a partial order. Now the formula

$$ab\|c \subseteq (a\|c)b + a(b\|c) \qquad \text{(ii)}$$

(cf. section 4.5.7) does hold in concurrent bisimulation semantics: in $ab\|c$ there are five possibilities: c occurs before a, simultaneous with a, between a and b, simultaneous with b, or after b. Each of these possibilities is already present at the right hand side, so the summand $ab\|c$ says nothing new and can be deleted. However, (ii) is incompatible with True concurrency: in $ab\|c$ the action c occurs independently of both a and b, while in $(a\|c)b+a(b\|c)$ the c action occurs either causally before b or causally behind a.

Hence the True concurrency approach depends in a subtle way on a philosophically complicated issue like causality (for a philosophical discussion of the concept of causality see HOSPERS [Ho]). However, it is also possible to reject formulas like (ii) on more earthly grounds, as for example real-time behaviour. If we suppose that the execution of an atomic action takes a fixed amount of time (or that a fixed amount of time elapses between the instantaneous occurrence of an atomic action and the following action), and if we furthermore assume that c takes as much time as a and b together, then $ab\|c$ can be executed much faster than $(a\|c)b+a(b\|c)$.

This section is devoted to the causal approach to True concurrency. In the next section the real-time approach will be examined. There we will see to what extent these approaches coincide. After some preliminaries we start with the definition of the pomset equivalence, a partial order approach stemming from PRATT [Pr1-2], which is a variant of trace theory as originated by MAZURKIEWICZ (see [M1-2] and [AR]).

5.1. Causality.

5.1.1. DEFINITION: An *A-labelled partial ordered set* is a triple $(X,<,l)$ with X a set, $<$ a partial order on X, and $l: X \to A$ a labelling function. Two such sets $(X_0,<_0,l_0)$ and $(X_1,<_1,l_1)$ are *isomorphic* if there exists a bijective mapping $f: X_0 \to X_1$ such that $f(x)<f(y) \Leftrightarrow x<y$ and $l_1(f(x))=l_0(x)$.

A *partial ordered multiset (pomset)* over A is an isomorphy class of A-labelled partial ordered sets. As usual, pomsets can be made setlike by requiring that the elements of the sets X should be chosen from a given set.

A *totally ordered multiset (tomset)* over A can be defined similarly. There exists a 1-1 correspondence between sequences $\sigma \in A^*$ and finite tomsets. Let $\hat{\sigma}$ denote the tomset corresponding with σ. A sequence $\sigma \in A^*$ is a *sequentialisation* of a pomset ρ, if $\hat{\sigma}$ can be obtained by expanding the partial order of ρ into a total one.

5.1.2. DEFINITION: Let $N=(S,T,F,M_{in},l) \in \mathbf{N}_\lambda(A)$. A transition t' is *directly preceded* by a transition t if $t^\bullet \cap {}^\bullet t' \neq \emptyset$. For $\alpha = t_1 * \cdots * t_m \in T^*$ define the relation $<$ on the set $Oc(\alpha) = \{(t_i,i) \mid 1 \leq i \leq m\}$ of numbered occurrences of transitions in α as the partial order generated by: $(t_i,i)<(t_j,j)$ if $i<j$ and t_j is directly preceded by t_j. This makes $(Oc(\alpha),<,l')$ with $l'(t_i,i)=l(t_i)$ into an A-labelled partial ordered set. The corresponding pomset is denoted by $pom(\alpha)$.

Let M,M' be markings of N, $\alpha \in T^*$, $\rho = pom(\alpha)$ and let $M[\alpha>M'$. In this case we say that ρ is *M-enabled*, notation $M[\rho>$. We also say that *M' is obtained from M by firing* ρ, notation $M[\rho>M'$.

5.1.3. PROPOSITION: Let $N=(S,T,F,M_{in},l) \in \mathbf{N}_\lambda(A)$, let M,M' be markings of N, and let ρ be a pomset over A such that $M[\rho>M'$. Let $\sigma \in A^*$ be a sequentialisation of ρ. Then $M[\sigma>M'$ (using definition 2.3.(v)). Note that we cannot conclude $M[\hat{\sigma}>M'$.

5.2. Pomset semantics.

5.2.1. DEFINITION: The set $pom(N)$ of pomsets (with termination information) of a net $N \in \mathbf{N}_\lambda(A)$ is given by $pom(N) = \{pom(\alpha) \mid M_{in}[\alpha>\} \cup \{(pom(\alpha),\surd) \mid M_{in}[\alpha> \emptyset\}$. Two nets $N_1,N_2 \in \mathbf{N}_\lambda(A)$ are *pomset equivalent*, notation $N_1 \equiv_{pom} N_2$, if $pom(N_1)=pom(N_2)$.

5.2.2. EXAMPLES: (Note that the first two nets *are* identified in concurrent bisimulation semantics.)

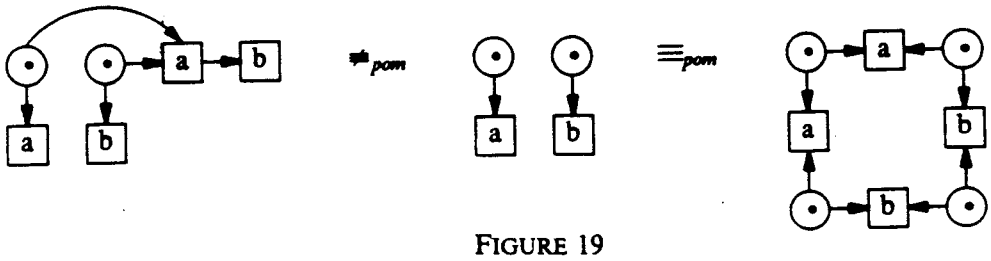


FIGURE 19

5.2.3. PROPOSITION: *$\equiv_{pom}$ is an equivalence relation on $\mathbb{N}_\lambda(A)$, which is a congruence w.r.t. the operators of section 1.*
5.2.4. PROPOSITION: *$\mathbb{N}_\lambda(A)/\equiv_{pom}$ satisfies the axioms of table 1.*
5.2.5. PROPOSITION: $\equiv_{occ} \Rightarrow \equiv_{pom}$.

5.3. Pomset bisimulation semantics.
Although the partial order approach sketched above preserves causality, other features of concurrency get lost. The equation

$$a(b+c) = ab+ac \qquad \text{(iii)}$$

is not valid in (concurrent) bisimulation semantics, since the timing of the choice between b and c is different on the left and right hand side. However, (iii) is satisfied in $\mathbb{N}_\lambda(A)/\equiv_{pom}$, so in this model all information about the timing of choices is lost. As argued by (for instance) MILNER [Mi], this implies that deadlock behaviour cannot be described in the model. Thus we encountered bisimulation equivalences, respecting branching time but violating causality, and pomset equivalence which respects causality but violates branching time. Now the natural question arises whether the virtues of both approaches can be combined or not, i.e. if a model of concurrency exists respecting both causality and branching time. Such a model was presented by BOUDOL & CASTELLANI [BC]. A variant of this model, adapted to the setting of the present paper will be treated below (Boudol and Castellani used a different language, and event structures instead of Petri nets).

5.3.1. DEFINITION: Let $N_0,N_1 \in \mathbb{N}_\lambda(A)$. A relation $R \subseteq Pow(S_0)\times Pow(S_1)$ is a *pomset bisimulation* between N_0 and N_1, notation $R:N_0 \leftrightarrows_{pom} N_1$, if:
1. $(M_{in})_0\ R\ (M_{in})_1$;
2. if $M_0\ R\ M_1$ and for $\alpha_0 \in T_0^*$ $M_0[\alpha_0>M'_0$, then there are $\alpha_1 \in T_1^*$ and $M'_1 \subseteq S_1$ such that $pom(\alpha_0)=pom(\alpha_1)$, $M_1[\alpha_1>M'_1$ and $M'_0\ R\ M'_1$;
3. as 2 but with the roles of N_0 and N_1 reversed;
4. $M_0\ R\ M_1 \Rightarrow (M_0=\emptyset \Leftrightarrow M_1=\emptyset)$.

N_0 and N_1 are *pomset bisimilar*, notation $N_0 \leftrightarrows_{pom} N_1$, iff there is an $R:N_0 \leftrightarrows_{pom} N_1$.

5.3.2. EXAMPLES:

$$a(b\|c)+(a\|c)b \not\leftrightarrows_{pom} a(b\|c)+(a\|c)b+ab\|c \quad \text{(as for } \equiv_{pom}\text{, in contrast with } \leftrightarrows_c\text{, see (ii) in §5.0)} \qquad \text{(a)}$$

$$a\|b+ab \not\leftrightarrows_{pom} a\|b \leftrightarrows_{pom} a\|b+a\|b \quad \text{(as for } \equiv_{pom}\text{, see figure 19)} \qquad \text{(b)}$$

$$a\|(b+c)+a\|b+(a+c)\|b \leftrightarrows_{pom} a\|(b+c)+(a+c)\|b \quad \text{(taken from [BC])} \qquad \text{(c)}$$

5.3.3. PROPOSITION: *$\leftrightarrows_{pom}$ is an equivalence relation on $\mathbb{N}_\lambda(A)$, which is a congruence w.r.t. the operators of section 1.*
5.3.4. PROPOSITION: *$\mathbb{N}_\lambda(A)/\leftrightarrows_{pom}$ satisfies the axioms of table 1.*
5.3.5. PROPOSITION: $\equiv_{occ} \Rightarrow \leftrightarrows_{pom} \Rightarrow \leftrightarrows_c$, $\leftrightarrows$ and $\equiv_{pom}$.

5.4. Combining causality and branching time.
The formula (ii) respects branching time, but violates causality. It holds in concurrent bisimulation semantics, but does not hold in pomset or pomset bisimulation semantics. On the other hand the formula

$$ab\|c \subseteq a(b+d)\|c \qquad \text{(iv)}$$

respects causality but violates branching time. It holds in pomset semantics, but not in (concurrent) bisimulation or pomset bisimulation semantics. The combination of (ii) and (iv):

$$ab\|c \subseteq a(b+d)\|c+(a\|c)b+a(b\|c) \qquad \text{(v)}$$

respects both causality and branching time. However, it violates the idea behind the combination of these two concepts. The right hand side will perform an a and a c action, and either a b or a d. The action c can occur causally before b or causally behind a. It is also possible that c occurs independent of both a and b, but in that case c occurs also independent of the choice between b and d. The summand $ab\|c$ adds the possibility that c occurs independent of both a and b, but causally behind the choice in favour of b. Hence (v) violates the real combination of causality and branching time. Nevertheless it is satisfied in pomset bisimulation semantics. The model based on the following *generalised pomset bisimulation equivalence* does not have this disadvantage. However, in [BC], BOUDOL & CASTELLANI presented a proof system for a language without communication, that is sound and complete for closed terms with respect to pomset bisimulation semantics. Such a result is not available for the generalised version.

5.4.1. DEFINITION: Let $N_0,N_1 \in \mathbb{N}_\lambda(A)$. A relation $R \subseteq Pow(S_0)\times Pow(S_1)$ is a *generalised pomset bisimulation* between N_0 and N_1, notation $R:N_0 \leftrightarrows_{gpom} N_1$, if:
1. $(M_{in})_0\ R\ (M_{in})_1$;
2. if $M_0\ R\ M_1$ and for $\alpha_0 = t_{01}*t_{02}*\cdots*t_{0n} \in T_0^*$, $M_0\ [t_{01}>M_{01}\ [t_{02}>M_{02}\cdots\ [t_{0n}>M_{0n}$, then there are α_1 =

$t_{11}*t_{12}*\cdots*t_{1n}\in T_1$ such that $pom(\alpha_0)=pom(\alpha_1)$, $M_1\ [t_{11}>M_{11}\ [t_{12}>M_{12}\cdots\ [t_{1n}>M_{1n}$, and $M_{0i}\ R\ M_{1i}$ for $1\leq i\leq n$;

3. as 2 but with the roles of N_0 and N_1 reversed;
4. $M_0\ R\ M_1\ \Rightarrow\ (M_0=\emptyset\Leftrightarrow M_1=\emptyset)$.

N_0 and N_1 are *generalised pomset bisimilar*, notation $N_0\leftrightarrows_{gpom}N_1$, iff there is an $R:N_0\leftrightarrows_{gpom}N_1$.

5.4.2. EXAMPLES:

$$ab\|c+a(b+d)\|c+(a\|c)b+a(b\|c)\ \not\leftrightarrows_{gpom}\ a(b+d)\|c+(a\|c)b+a(b\|c)\quad\text{(unlike }\leftrightarrows_{pom}\text{, see (v))}\qquad\text{(a)}$$

$$a\|(b+c)+a\|b+(a+c)\|b\ \leftrightarrows_{gpom}\ a\|(b+c)+(a+c)\|b\quad\text{(as for }\leftrightarrows_{pom}\text{, see 5.3.2(c))}\qquad\text{(b)}$$

5.4.3. PROPOSITION: *$\leftrightarrows_{gpom}$ is an equivalence relation on $\mathbb{N}_\lambda(A)$, which is a congruence w.r.t. the operators of section 1.*
5.4.4. PROPOSITION: *$\mathbb{N}_\lambda(A)/\leftrightarrows_{gpom}$ satisfies the axioms of table 1.*
5.4.5. PROPOSITION: $\equiv_{occ}\ \Rightarrow\ \leftrightarrows_{gpom}\ \Rightarrow\ \leftrightarrows_{pom}$.

§6 REAL-TIME SEMANTICS

In this section we will describe a possible real-time interpretation of process algebra. First we show how real-time behaviour can be attached to Petri nets. This gives the timed Petri net model as presented in, for example, CARLIER, CHRETIENNE & GIRAULT [CCG]. In BAETEN, BERGSTRA & KLOP [BBK2] it is shown how operational rewrite systems can be used to give real-time semantics for the ACP language. The intuition behind this semantics is comparable with the intuition behind the timed Petri nets. New in our approach is the notion of *real-time consistency*. An equivalence relation on Petri nets is real-time consistent if it does not identify nets with a different real-time behaviour. A model based on a real-time consistent equivalence makes it possible to reason about concurrent systems in a real-time consistent manner without having to deal with the full complexity of real-time. We show that the concurrent and pomset bisimulation equivalences are not real-time consistent. However, in the next section we will present an equivalence relation on nets which *is* real-time consistent.

6.1. Timed Petri nets. Let $N=(S,T,F,M_{in},l)$ be a safe labelled marked net. Time is introduced in the following manner. We assume the presence of a function $\tau:A\rightarrow\mathbb{R}^+$. The τ-function associates a (fixed) processing time with each atomic action. Here we have chosen the range of τ to be the set $\mathbb{R}^+$, but this could also have been $\mathbb{Q}^+$ or $\mathbb{N}^+$. We assume that τ has a positive lower bound in order to avoid Zeno's paradox. Further we restrict ourselves to nets in $\mathbb{N}_\lambda(A)$ (so we have bounded parallelism). As a consequence of these assumptions the set of points in time at which some transition starts or ends firing, will be discrete. In case a transition u fires at time t, it removes the tokens from the input places. $\tau(l(u))$ time units later tokens are placed in the output places of u. We assume that it takes no time to resolve conflicts: when a transition can fire, it will fire immediately, or it will be disabled immediately. This means that we have maximal parallelism. We make these intuitions formal in the following definitions:

6.2. DEFINITION: Let $N=(S,T,F,M_{in},l)\in\mathbb{N}_\lambda(A)$ and let $\tau:A\rightarrow\mathbb{R}^+$. An *instantaneous description* is a 4-tuple (M,U,ρ,t) where:

- $M\subseteq S$ is the set of places with a token;
- $U\subseteq T$ is the set of transitions which are firing. If a transition is firing, its input places are already empty $(\forall u\in U:{}^\bullet u\cap M=\emptyset)$, whereas its output places are still empty;
- $\rho:T\rightarrow[0,\infty)$ is a function which gives for each transition the residual processing time. $\rho(u)=0$ for those transitions which are not firing;
- $t\in[0,\infty)$ is the time.

We will now define a transition relation $\rightarrow$, which tells how one instantaneous description can evolve into another.

6.3. DEFINITION: Let $N=(S,T,F,M_{in},l)\in\mathbb{N}_\lambda(A)$, $\tau:A\rightarrow\mathbb{R}^+$ and let (M,U,ρ,l) be an instantaneous description of N. The binary relation $\rightarrow$ is defined by:

1. if $M[u>$, $\rho'(u)=\tau(l(u))$ and $\rho'(\overline{u})=\rho(\overline{u})$ if $\overline{u}\neq u$, then: $(M,U,\rho,t)\rightarrow(M-{}^\bullet u,U\cup\{u\},\rho',t)$;
2. if $u\in U$ and $\rho(u)=0$, then: $(M,U,\rho,t)\rightarrow(M\cup u^\bullet,U-\{u\},\rho,t)$;
3. if $\forall u\in T:\neg M[u>$, $U\neq\emptyset$, $0<\Delta t=\min\{\rho(u)\mid u\in U\}$, $\rho'(u)=\rho(u)-\Delta t$ if $u\in U$, and $\rho'(u)=0$ otherwise, then: $(M,U,\rho,t)\rightarrow(M,U,\rho',t+\Delta t)$.

6.4. DEFINITION: Let $N\in\mathbb{N}_\lambda(A)$. A *real-time execution* of N is a sequence $<I_i>_{i<n}$ of instantaneous descriptions of N such that $I_0=(M_{in},\emptyset,\lambda x.0,0)$ and $I_i\rightarrow I_{i+1}$ for $i<n$.

The following definition is derived from REED & ROSCOE [RR].

6.5. DEFINITION: A *timed action* is an ordered pair (t,a), where a is an atomic action and $t \in [0,\infty)$ is the time at which it occurs. The set $[0,\infty) \times A$ of all timed actions is denoted TA. The set of *timed traces* is:

$$(TA)^{\bullet}_{\leq} = \{\sigma \in TA^* \mid \text{if } (t,a) \text{ precedes } (t',a') \text{ in } \sigma, \text{ then } t \leq t'\}$$

6.6. In a trivial way we can associate a timed trace with each real-time execution of a net $N \in \mathbf{N}_\lambda(A)$: we take the sequence of actions executed by the system, together with their starting times. Let $tt_\tau : \mathbf{N}_\lambda(A) \to (TA)^{\bullet}_{\leq}$ be the function which, for given $\tau : A \to \mathbf{R}^+$, associates with each net the set of its timed traces.

6.7. DEFINITION: An equivalence relation $\equiv$ on $\mathbf{N}_\lambda(A)$ is called *real-time consistent*, if for all $\tau : A \to \mathbf{R}^+$ with the property $\exists \epsilon > 0\ \forall a \in A$: $\tau(a) > \epsilon$, and for all $N_0, N_1 \in \mathbf{N}_\lambda(A)$: $N_0 \equiv N_1 \Rightarrow tt_\tau(N_0) = tt_\tau(N_1)$.

6.8. PROPOSITION: *The equivalence relation* $\leftrightarrows_{gpom}$ *is not real-time consistent.*
PROOF: Below we give two nets which are generalised pomset bisimilar, but have different timed traces.

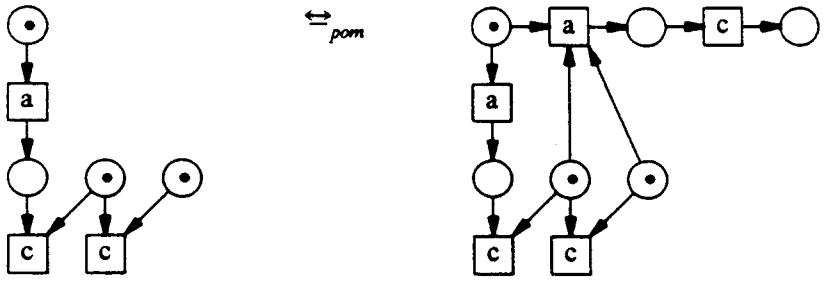


FIGURE 20

If we define communication by $\gamma(r,s) = c$ and if $H = \{r,s\}$, then we can express the identity algebraically as follows:

$$\partial_H(ar \| r \| s) = \partial_H(ar \| r \| s) + ac\delta$$

Assume $\tau(a) = \tau(c) = 1$. Now any real-time execution of the net on the left will start with an a and a c action. The net on the right however, has also the possibility to start with an a action only, followed by a c action after 1 time unit.

6.9. COROLLARY: *The equivalence relations* $\leftrightarrows_{pom}$, $\leftrightarrows_{c}$, $\leftrightarrows$ *and* $\equiv_{pom}$ *are not real-time consistent.*

6.10. QUESTION: Is $\leftrightarrows_{gpom}$ real-time consistent for fully observable process expressions, as defined in section 7.5.2?

§7 ST-BISIMULATION SEMANTICS

In this section we present a non-interleaved Petri net model for the theory of section 1, together with a complete proof system for closed process expressions without autoparallelism (cf. section 7.3). The model is based on a real-time consistent equivalence on nets, called *ST-bisimulation equivalence*. The idea behind ST-bisimulation equivalence is rather simple: A bisimulation can be viewed as a relation between the states of two systems. In the graph model of [BBK1], the states are the nodes in the graphs. In the bisimulation equivalence on Petri nets which we presented in the previous sections, the states of the system are distributed entities: namely the set of places containing a token. In the philosophy leading to the ST-bisimulation the state of a system is the set of places containing a token, together with the sequence of transitions which are firing, in the order they started firing. This is just what one obtains when leaving out the real-time information from the instantaneous descriptions as defined in section 6.2 (the order of the transitions can be derived from the function ρ in 6.2).

7.1.1. DEFINITION: Let $N_0, N_1 \in \mathbf{N}_\lambda(A)$. A relation $R \subseteq (Pow(S_0) \times T_0^{\bullet}) \times (Pow(S_1) \times T_1^{\bullet})$ is an *ST-bisimulation* between N_0 and N_1, notation $R : N_0 \leftrightarrows_{ST} N_1$, if:

1. $((M_{in})_0, \epsilon)\ R\ ((M_{in})_1, \epsilon)$;
2. if $(M_0, \sigma_0)\ R\ (M_1, \sigma_1)$ and $M_0[t_0\rangle$, then there is a $t_1 \in T_1$ such that $l_0(t_0) = l_1(t_1)$, $M_1[t_1\rangle$ and $(M_0 -^{\bullet} t_0\ ,\ \sigma_0 {}^{\bullet}t_0)\ R\ (M_1 -^{\bullet} t_1\ ,\ \sigma_1 {}^{\bullet}t_1)$;
3. as 2 but with the roles of N_0 and N_1 reversed;
4. if $(M_0\ ,\ \sigma_0 {}^{\bullet}t_0 {}^{\bullet}\rho_0)\ R\ (M_1\ ,\ \sigma_1 {}^{\bullet}t_1 {}^{\bullet}\rho_1)$ and $|\sigma_0| = |\sigma_1|$, then $(M_0 \cup t_0^{\bullet}\ ,\ \sigma_0 {}^{\bullet}\rho_0)\ R\ (M_1 \cup t_1^{\bullet}\ ,\ \sigma_1 {}^{\bullet}\rho_1)$;
5. as 4 but with the roles of N_0 and N_1 reversed;
6. $(M_0, \sigma_0)\ R\ (M_1, \sigma_1) \Rightarrow ((M_0, \sigma_0) = (\varnothing, \epsilon) \Leftrightarrow (M_1, \sigma_1) = (\varnothing, \epsilon))$.

N_0 and N_1 are *ST-bisimilar*, notation $N_0 \leftrightarrows_{ST} N_1$, iff there is an $R : N_0 \leftrightarrows_{ST} N_1$.

7.1.2. EXAMPLES: Define communication by $\gamma(r,s) = c$ and let $H = \{r,s\}$. Then:

$\partial_H(ar\|r\|s) + ac\delta \not\leftrightarrows_{ST} \partial_H(ar\|r\|s) \leftrightarrows_{ST} (a\|c)\delta$ (both parts unlike $\leftrightarrows_{pom}$ and $\leftrightarrows_{gpom}$, see §6.8) (a)

$ab\|c + a(b+d)\|c + (a\|c)b + a(b\|c) \not\leftrightarrows_{ST} a(b+d)\|c + (a\|c)b + a(b\|c)$ (unlike $\leftrightarrows_{pom}$ see §5.4) (b)

$a\|(b+c) + a\|b + (a+c)\|b \leftrightarrows_{ST} a\|(b+c) + (a+c)\|b$ (as for $\leftrightarrows_{pom}$ and $\leftrightarrows_{gpom}$ see example 5.3.2(c)) (c)

The second part of example (a) shows that ST-bisimulation does not respect causality. Both parts together show that pomset bisimulation and ST-bisimulation are incomparable.

7.1.3. PROPOSITION: *$\leftrightarrows_{ST}$ is an equivalence relation on $\mathbf{N}_\lambda(A)$, which is a congruence w.r.t. the operators of section 1.*
7.1.4. PROPOSITION: *$\mathbf{N}_\lambda(A)/\leftrightarrows_{ST}$ satisfies the axioms of table 1.*
7.1.5. PROPOSITION: *All implications between the equivalences of this paper are displayed below:*

$$\begin{array}{ccccccc} & \nearrow & \leftrightarrows_{gpom} & \Rightarrow & \leftrightarrows_{pom} & \Rightarrow & \equiv_{pom} \\ \equiv_{occ} & & & & \Downarrow & & \\ & \searrow & \leftrightarrows_{ST} & \Rightarrow & \leftrightarrows_{c} & \Rightarrow & \leftrightarrows \end{array}$$

7.2. THEOREM: *$\leftrightarrows_{ST}$ is real-time consistent.*

7.3. A complete proof system for closed terms without autoparallelism.
A net is said to have *autoparallelism* if two transitions with the same label can fire concurrently (from some reachable marking). A closed process expression has autoparallelism if this is the case for the associated net. Next we will present a proof system for ST-bisimulation semantics which is sound and complete for closed terms without autoparallelism. The structure of this proof system is as follows: We introduce an operator *split* that splits any atomic action a into the sequential composition of actions a^+ and a^-, representing the beginning and the end of a. Then we prove that $N_0 \leftrightarrows_{ST} N_1$ iff $split(N_0) \leftrightarrows split(N_1)$. Now we already have a proof system for $\leftrightarrows$, which is sound and complete for closed terms, namely ACP. Thus, a proof system for $\leftrightarrows_{ST}$ can be obtained from ACP, by adding a sound set of axioms for the *split* operator that allows any closed term $split(t)$ to be rewritten into a closed ACP-term (without occurrences of the *split* operator). In this way the interleaving based axiom system ACP can be used to prove identities in the non-interleaved ST-bisimulation semantics.

7.3.1. Syntax. Formally, this idea will be implemented by a two-sorted algebra. We have a sort P of processes (the processes in a domain with ST-bisimulation semantics) and an auxiliary sort SP of split processes, in a domain with interleaving. On P we have the constants and operators of section 1; on SP we have constants δ, a^+, a^- (for $a \in A$), the operators $+$, $\cdot$, $\|$ and δ_H of section 1, and the auxiliary operators $\lfloor\!\lfloor$ and $|$ of section 4.4. Furthermore there is a unary operator $split: P \to SP$.

7.3.2. Semantics. Let A be a set of atomic actions and $\gamma: A_\delta \times A_\delta \to A_\delta$ a communication function. We define the set of split actions $A^\pm$ and the communication function $\gamma: A_\delta^\pm \times A_\delta^\pm \to A_\delta^\pm$ by:

$$A^\pm = \{a^+, a^- \mid a \in A\} \qquad \gamma(a^+, b^+) = \gamma(a,b)^+ \qquad \gamma(a^-, b^-) = \gamma(a,b)^- \qquad \gamma(a^+, b^-) = \delta$$

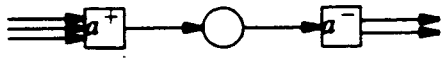
Now we give a model for this two-sorted signature. The set $\mathbf{N}_\lambda(A)/\leftrightarrows_{ST}$ will be the domain of processes and $\mathbf{N}_\lambda(A^\pm)/\leftrightarrows$ will be the domain of split processes. In order to define the operator $split: \mathbf{N}_\lambda(A)/\leftrightarrows_{ST} \to \mathbf{N}_\lambda(A^\pm)/\leftrightarrows$ we first define $split: \mathbf{N}_\lambda(A) \to \mathbf{N}_\lambda(A^\pm)$ and then check that the pair $(\leftrightarrows_{ST}, \leftrightarrows)$ is a congruence for this operator. The remaining constants and operators are defined as before.

7.3.2.1. *split*: Let $N_0 \in \mathbf{N}_\lambda(A)$. $N_1 = split(N_0) \in \mathbf{N}_\lambda(A^\pm)$ is defined by:

$$S_1 = S_0 \dot\cup T_0$$
$$T_1 = \{t^+, t^- \mid t \in T_0\}$$
$$F_1 = \{(s,t^+) \mid (s,t) \in F_0\} \dot\cup \{(t^+,t) \mid t \in T_0\} \dot\cup \{(t,t^-) \mid t \in T_0\} \dot\cup \{(t^-,s) \mid (t,s) \in F_0\}$$
$$(M_{in})_1 = (M_{in})_0$$
$$l_1(t^+) = (l_2(t))^+ \text{ and } l_1(t^-) = (l_2(t))^-$$

So $split(N)$ is obtained from N, by replacing any net segment

by a net segment



That $(\leftrightarrows_{ST}, \leftrightarrows)$ is a congruence for *split*, follows from the following proposition.

7.3.3. PROPOSITION: *For any $N_0, N_1 \in \mathbf{N}_\lambda(A)$: $N_0 \leftrightarrows_{ST} N_1 \Rightarrow split(N_0) \leftrightarrows split(N_1)$.*

7.3.4. PROPOSITION: *For any* $N_0,N_1 \in \mathbb{N}_\lambda(A)$ *without autoparallelism:* $split(N_0) \leftrightarrows split(N_1) \Rightarrow N_0 \leftrightarrows_{ST} N_1$.

7.3.5. The split rule. Let SPR be the rule $split(x)=split(y) \Rightarrow x=y$. Proposition 7.3.4 states its soundness for processes without autoparallelism. Furthermore proposition 4.4.6 states the soundness of ACP on sort *SP*. Now let SPLIT be a sound set of axioms, such that for any closed process expression p of sort P there is a closed ACP-expression q such that SPLIT $\vdash split(p)=q$. Then it follows trivially that SPLIT+ACP+SPR is a sound and complete proof system for closed terms without autoparallelism. So it suffices to find a suitable version of SPLIT. This we will do in three stages.

7.4. The case without communication. Suppose that $\gamma(a,b)=\delta$ for $a,b \in A$. Let SP^* be the theory presented in the left upper block of table 3, but with SP3 replaced by SP3*: $split(x \| y)=split(x) \| split(y)$. Then SP^* is sound with respect to the model of section 7.3.2 and for any closed process expression p of sort P there is a closed ACP-expression q of sort SP such that $SP^* \vdash split(p)=q$.

7.5. The case with fully observable processes. If we just expand the approach of section 7.4. with communication as defined in 7.3.2, then a serious problem arises: SP3* is not sound any more. Counterexample (if $\gamma(a,b)=c$):

$$split(a \| b) = a^+(a^-b^+b^- + b^+(a^-b^- + b^-a^-)) + b^+(b^-a^+a^- + a^+(b^-a^- + a^-b^-)) + c^+c^-$$

$$split(a) \| split(b) = a^+(a^-b^+b^- + b^+(a^-b^- + b^-a^- + c^-)) +$$
$$+ b^+(b^-a^+a^- + a^+(b^-a^- + a^-b^- + c^-)) + c^+(b^-a^- + a^-b^- + c^-)$$

Note that $split(a) \| split(b)$ is a very unrealistic process since certain actions can end before they begin. It turns out that sometimes a^+ communicates with b^+, while a^- and b^- occur independently. In order to disable this suspicious behaviour we will introduce a state operator $\lambda_\varnothing$, that renames actions a^- into δ if they are not preceded by an action a^+. Then we can replace axiom SP3* by SP3 (see table 3).

For the general theory of state operators see BAETEN & BERGSTRA [BB]. Our state operator remembers which actions are currently firing. This memory is located in a subscript S, with S ranging over $Pow(A)$. So we add operators $\lambda_S: SP \to SP$ to our signature for $S \subseteq A$. The axioms for λ_S are presented in the right-hand half of table 3.

$split(x+y) = split(x)+split(y)$	SP1	$\lambda_S(\delta) = \delta$	L1
$split(xy) = split(x)\cdot split(y)$	SP2	$\lambda_S(a^+) = a^+$	L2
$split(x \| y) = \lambda_\varnothing(split(x) \| split(y))$	SP3	$\lambda_S(a^-) = a^-$ if $a \in S$	L3
$split(\partial_H(x)) = \partial_{H^=}(split(y))$	SP4	$\lambda_S(a^-) = \delta$ if $a \notin S$	L4
$split(\delta) = \delta$	SP5	$\lambda_S(a^+x) = a^+ \cdot \lambda_{S \cup \{a\}}(x)$	L5
$split(a) = a^+ \cdot a^-$	SP6	$\lambda_S(a^-x) = a^- \cdot \lambda_{S-\{a\}}(x)$ if $a \in S$	L6
		$\lambda_S(a^-x) = \delta$ if $a \notin S$	L7
$split(x)=split(y) \Rightarrow x=y$	SPR	$\lambda_S(x+y) = \lambda_S(x)+\lambda_S(y)$	L8

TABLE 3

Now SP+L allows any closed process expression $split(p)$ with p of sort P to be rewritten in a closed ACP-term. However, the axiom SP3 is still not sound for all processes. Therefore we will first limit ourselves to a restricted domain of processes, for which its soundness can be proved.

7.5.1. DEFINITION: The alphabet $\alpha(N)$ of a net N is the set of labels of transitions which are M-enabled from a marking $M \in [M_{in}>$. Remark that if $N_0 \leftrightarrows N_1$ then $\alpha(N_0) = \alpha(N_1)$. Now the alphabet $\alpha(p)$ of a closed process expression p is the alphabet of the associated net.

7.5.2. DEFINITION: A closed process expression p is *fully observable* if for any subexpression $x \| y$ of p we have $\alpha(x) \cap \alpha(y) = \varnothing$, and for any $a,c \in \alpha(x)$, $b,d \in \alpha(y)$ we have:

$$a \neq \gamma(a,b) \neq b$$

$$\gamma(a,b) = \gamma(c,d) \neq \delta \Rightarrow a=c \wedge b=d$$

In this case any action can be traced back to the components in a merge it originated from. Obviously a fully observable process expression has no autoparallelism.

7.5.3. PROPOSITION: *The axiom SP3 holds for fully observable closed process expressions* $u \| v$. *Hence* SP+ACP+L+SPR *is sound and complete for fully observable closed process expressions.*

7.6. The general case, but still without autoparallelism. This case can be derived from the previous one by first doing some 'preprocessing' to make closed terms fully observable. For details see our full paper [GV].

7.7. CONJECTURE: We can make the proof system sound and complete for processes in which at most n transitions with the same label can occur concurrently by splitting atomic actions into the sequential composition of $n+1$ parts.

REFERENCES

[AR] AALBERSBERG, IJ.J. & G. ROZENBERG, *Theory of traces,* Technical Report No. 86-16, Institute of Applied Mathematics and Computer Science, University of Leiden, 1986.

[BB] BAETEN, J.C.M. & J.A. BERGSTRA, *Global renaming operators in concrete process algebra,* CWI Report CS-R8521, Amsterdam, 1985.

[BBK1] BAETEN, J.C.M., J.A. BERGSTRA & J.W. KLOP, *On the consistency of Koomen's fair abstraction rule,* CWI Report CS-R8511, Amsterdam, 1985, to appear in Theor. Comp. Sci.

[BBK2] BAETEN, J.C.M., J.A. BERGSTRA & J.W. KLOP, *An operational semantics for process algebra,* CWI Report CS-R8522, Amsterdam, 1985, to appear in: Proc. Banach semester, Warschau 1985, North-Holland.

[BK1] BERGSTRA, J.A. & J.W. KLOP, *Process algebra for synchronous communication,* Information & Control 60 (1/3), 1984, pp. 109-137.

[BK2] BERGSTRA, J.A. & J.W. KLOP, *Process Algebra: Specification and Verification in Bisimulation Semantics,* In: Proc. CWI Symposium Math. & Comp. Sci. (M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens, eds.), North Holland, 1986, pp. 61-94.

[BK3] BERGSTRA, J.A. & J.W. KLOP, *Algebra of Communicating processes with abstraction,* Theor. Comp. Sci. 37(1), 1985, pp. 77-121.

[BC] BOUDOL, G. & I. CASTELLANI, *On the semantics of concurrency: partial orders and transition systems,* Rapports de Recherche No 550, INRIA, Centre Sophia Antipolis, 1986.

[CCG] CARLIER, J., CHRETIENNE & C. GIRAULT, *Modelling scheduling problems with timed Petri nets,* In: Advances in Petri Nets 1984 (G. Rozenberg, ed.), Springer LNCS 188, 1985, pp. 62-82.

[GV] VAN GLABBEEK, R.J. & F.W. VAANDRAGER, *Petri net models for algebraic theories of concurrency,* to appear as: CWI Report CS-R87.., Amsterdam, 1987.

[G] GOLTZ, U., *Building Structured Petri Nets,* Arbeitspapiere der GMD 223, Sankt Augustin, 1986.

[GM] GOLTZ, U. & A. MYCROFT, *On the relationship of CCS and Petri nets,* In: Proc. ICALP 84 (J. Paredaens, ed.), Springer LNCS 172, 1984, pp. 196-208.

[Ho] HOSPERS, J., *An Introduction to Philosophical Analysis,* second edition, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[M1] MAZURKIEWICZ, A., *Concurrent program schemes and their interpretations,* Report DAIMI PB-78, Computer Science Department, Aarhus University, Aarhus, 1978.

[M2] MAZURKIEWICZ, A., *Semantics of concurrent systems: a modular fixed-point trace approach,* In: Advances in Petri Nets 1984 (G. Rozenberg, ed.), Springer LNCS 188, 1985, pp. 353-375.

[Mi] MILNER, R., *A calculus for Communicating Systems,* Springer LNCS 92, 1980.

[NPW] NIELSEN, M., G.D. PLOTKIN & G. WINSKEL, *Petri nets, event structures and domains, part I.* Theor. Comp. Sci., 13(1). 1981, pp. 85-108.

[NT] NIELSEN, M. & P.S. THIAGARAJAN, *Degrees of Non-Determinism and Concurrency: A Petri Net View,* In: Proc. of the 5th Conf. on Found. of Softw. Techn. and Theor. Comp. Sci. (M. Joseph & R. Shyamasundar, eds.), Springer LNCS 181, 1984, pp. 89-118.

[Pa] PARK, D.M.R., *Concurrency and automata on infinite sequences,* Proc. 5th GI Conference (P. Deussen, ed.), Springer LNCS 104, 1981, pp. 167-183.

[Pe] PETRI, C.A., *Kommunikation mit Automaten,* Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[Po] POMELLO, L., *Some equivalence notions for concurrent systems. An overview.* In: Advances in Petri Nets 1985 (G. Rozenberg, ed.), Springer LNCS 222, 1986, pp. 381-400.

[Pr1] PRATT, V.R., *On the Composition of Processes,* Proc. of the 9th POPL, 1982, pp. 213-223.

[Pr2] PRATT, V.R., *Modelling Concurrency with Partial Orders,* International Journal of Parallel Programming, Vol. 15, No. 1, 1986, pp. 33-71.

[RR] REED, G.M. & A.W. ROSCOE, *A Timed Model for Communicating Sequential Processes,* In: Proc. ICALP 86 (L. Kott, ed.), Springer LNCS 226, 1986, pp. 314-323.

[R] REISIG, W., *Petri Nets, An Introduction,* EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.

[RT] ROZENBERG, G. & P.S. THIAGARAJAN, *Petri nets: basic notions, structure, behaviour.* In: Current Trends in Concurrency, Overviews and Tutorials (J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds.), Springer LNCS 224, 1986, pp. 585-668.

[W1] WINSKEL, G., *Event structure semantics for CCS and related languages,* In: Proc. 9th ICALP (M. Nielsen & E.M. Schmidt, eds.), Springer LNCS 140, 1982, pp. 561-576.

[W2] WINSKEL, G., *A new definition of morphism on Petri net,* In: Proc. STACS 84 (M. Fontet, K. Mehlhorn, eds.), Springer LNCS 166, 1984, pp. 140-150.