Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

L. Cherkasova

A fully abstract model for concurrent nondeterministic processes based on posets with non-actions

Computer Science/Department of Software Technology Report CS-R9031 July

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

A Fully Abstract Model for Concurrent Nondeterministic Processes Based on Posets with Non-Actions

Ludmila Cherkasova
Institute of Informatics Systems
Siberian Division, Academy of Sciences
630090 Novosibirsk, USSR

Abstract

The aim is to propose a new model based on the posets with non-actions to describe the concurrent nondeterministic processes and investigate their properties.
The algebra $SAFP_2$ of structured finite processes with three main operations for specifying sequentiality, concurrency and nondeterminism is introduced. Denotational semantics based on posets with non-actions is constructed for processes of $SAFP_2$.
Full abstractness of denotational semantics w.r.t. observational equivalence $\approx_+$ and proposed operational Petri net semantics is established.

1980 Mathematics Subject Classification (Zentralblatt für Mathematik): 68B10.
1985 Mathematics Subject Classification (Mathematical Reviews): 68Q55.
1987 CR Classification Scheme (Computing Reviews): F.3.2.
Key Words & Phrases: theory of concurrency, denotational semantics, posets with non-actions, observational equivalence, full abstractness, Petri nets, occurrence nets, operational Petri net semantics.
Note: Partial support received from NFI Project "Research and Education in Concurrent Systems" (REX).

1 Introduction

During the last decade, theory of concurrency has been a subject of intensive research attempts to develop a correct and adequate model and paradigm for concurrent computations.

Report CS-R9031
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

At the moment, there is no shortage of the models for concurrency based on interleaving semantics [BHR84, Ho85, Mil80, Mil83] or intended to describe, so-called, "true" concurrency [BouCa87, NPW81, Pet77, Pr87]. However, we dare propose a new one: posets with non-actions. Concurrent process, the elements of which are partially ordered by precedence relation (reflecting causal dependences) can be explicitly represented by partially ordered set. Thus the behaviour of concurrent nondeterministic process (or system) can be described by a set of its ''pure" concurrent processes. But in this case, the information about nondeterminism between the elements of the initial process (or system) is lost. To represent (implicitly) the nondeterminism on the semantic level, we introduce some "negative" information about the process actions which have not been chosen to be performed in process behaviour (the idea, in some sense, is similar to failure semantics for TCSP). Thus we introduce a dual alphabet of the "negated" symbols for denoting "non-actions", i.e. the symbols which point to the fact that the correspondent actions do not occur in a process behaviour because among the alternative actions including these ones the other action occurs.

Algebras proposed for process specification could be (informally) called as: "descriptive" algebras and "analytical" ones. In the descriptive algebras, process specification provides a good insight into structural properties of designed concurrent systems. Analytical algebras contain sufficient support for verification of behavioural properties. The goal of this paper is to build the next span of the bridge over the gap between descriptive and analytical theories of concurrent processes. The first steps in this direction were made in the papers [Ch88, Ch89, ChK90].

In the paper, we propose the algebra $SAFP_2$ for specifying concurrent nondeterministic processes which can be used as a calculus for subclass of finite Petri nets.

The basic operations of this algebra are $||$ (concurrency), ; (precedence), $\triangledown$ (alternative) over the set α of atomic actions. The semantics of these operations could be defined in quite different ways.

If we consider these operations as operations for building the structures of finite Petri nets, then, the so–called, descriptive algebra AFP_o emerges. Thus the structures of finite Petri nets can be specified using descriptive algebra AFP_o. However, if we consider the same formula (specifying structure of Petri net) as a formula supplied by the semantics, proposed for algebra $SAFP_2$, then the same formula specifies the behaviour of the net.

Thus, we can use the same formula as for describing structure of Petri net as for specifying and verifying its behaviour. So, the "structure", "descriptive" algebra AFP_o can be provided with the "analytical" abilities of $SAFP_2$, and the "analytical" algebra $SAFP_2$ can be supplied with "descriptive" abilities of AFP_o to specify the process structure as well as process behaviour.

Semantics of process described by formula of $SAFP_2$ is defined as a set of partial orders extended with some additional "negative" information. We introduce the notion

of observational equivalence $\approx_+$ for processes of $SAFP_2$. Two processes P and Q are called observationally equivalent if their "positive" parts (i.e. "observable" partially ordered actions of underlying posets) coincides. Full abstractness of proposed denotational semantics for $SAFP_2$ w.r.t. $\approx_+$ is established, i.e. the processes observationally equivalent in every context (it means that their positive parts can not be distinguished by observation of partially ordered actions in any context) have the same denotational semantics.

The last part of the paper proposes operational Petri net semantics for $SAFP_2$. It follows closely to works of [DDM86, Ol88], but differs in details of transition rules for operations of $SAFP_2$. The history preceding the creation of operational net semantics is quite interesting and amazing. In the paper [Pr87], construction of operational semantics for true concurrence was supposed as impossible due to interleaving flavour of Plotkin's style for operational semantics. However, at the same time it has been done in the paper [DDM86].

Full abstractness of denotational semantics for processes of $SAFP_2$ w.r.t. proposed operational Petri net semantics is established.

Acknowledgments

I would like to thank Jaco De Bakker and his group for providing pleasant and productive scientific atmosphere during my stay at CWI. I'm also indebted to Frits Vaandrager for his help, friendly support and attention to this work. Alexander Ustimenko (my former master student of Novosibirsk University) contributed to this paper by very helpful discussions, questions and remarks. The final version of the paper has been accomplished during my lecturing at the Paderborn University (West Germany). So, I would like to thank also Prof. Lutz Priese for his helpful comments and Petra Scheike for her expert typing of this manuscript.

2 Denotational semantics

Some additional information about $SAFP_2$ process algebra, its denotational semantics and axiomatization can be found in [Ch89].

2.1 Syntax of $SAFP_2$

Let $\alpha = \{a, b, c, \ldots\}$ be a finite alphabet of *actions* symbols (the action basis of a process).

The actions are combined into a composite process by the operations of ; *("precedence")*, $\triangledown$ *("exclusive" or, "alternative")* and $\|$ *("concurrency")*.

Intuitively, the process $(a; b)$, at first, performs the action a and only after that it performs the action b. The process $(a \triangledown b)$ consists of two possible behaviours: if it chooses the performance of the action a then the action b does not occur, and vice versa. The formula $(a \,\|\, b)$ specifies the process in which the actions a and b occur concurrently.

In our approach, we suppose that *each action has its own unique name.* Thus, if we have a process P consisting of different subprocesses P_1 and P_2 such that an action symbol c occurs in both P_1 and P_2, then the performance of the action c in P should be synchronized by the performances of c in P_1 and P_2 simultaneously, i.e. the process P can perform the action c if and only if both subprocesses P_1 and P_2 are ready to perform the action c. For example, the process formula $P = (a\ ;\ c) \,\|\, (b\ ;\ c)$ specifies the process in which the actions a and b are performed concurrently and only after that (i.e. after that both actions a and b have been executed) the action c can be performed.

Thus, if some action x in one process needs the action y in another process for an actual execution (it is a typical situation for communicating processes) then it can be easily specified by means of SAFP_2 using the same action symbols for actions x and y. Such an approach allows us do not to restrict the number or communicating processes (in CCS, the communication operator is binary, i.e. the communication is possible between two processes only).

So, the set of process terms is defined by the following production system:

$$P := a|P\|Q|P;Q|P\triangledown Q$$

where $a \in \alpha$.

2.2 Denotational semantics for SAFP_2

Semantics of a process described by a formula of SAFP_2 will be defined as a set of partial orders. Thus, a process described by the formula $(a \triangledown b)$ will be characterized by two partial orders: the first one defines the process behaviour if the action a is chosen to be performed and the second one defines the process behaviour if the action b is chosen for execution.

We would like to have a more complete information about *nondeterminism* in process structure *at a semantic level* of partial order representation. We include additionally a "negative" information in our consideration and reasoning about defined processes. In particular, we would like to know which actions have not been chosen during the concrete process behaviour.

Thus, to denote the fact that the action a during some process functioning does not occur (because some alternative action to the action a is performed) we introduce the negated symbol $\overline{a}$ and call it the non–action $\overline{a}$.

So, the process $(a \bigtriangledown b)$ is characterized by the following behaviours (partial orders): in the first one, the action a occurs and the non–action $\overline{b}$ appears; in the second one, the action b occurs and, additionally, the non–action $\overline{a}$ appears.

Let $\overline{\alpha} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ be dual to α alphabet of symbols for "*non-actions*".

Thus, each partial order representing one of the possible process behaviours has an *"observable" part* and a *"unobservable" one.* The "observable" part consists of the process actions which have been performed during this process run. The "unobservable" part consists of the non–actions which have not been executed (have not been chosen) during this process functioning.

However, there exists another reason why some actions could not be performed during some process functioning.

Let us consider a process defined by the following formula

$$P = (a \parallel b) \parallel (a \bigtriangledown b).$$

This process specification consists of two subspecifications $P_1 = (a \parallel b)$ and $P_2 = (a \bigtriangledown b)$.

The formula $P_1 = (a \parallel b)$ specifies a process in which both actions a and b should be performed and performed concurrently.

The formula $P_2 = (a \bigtriangledown b)$ defines two possible process functionings: either

1) the action a occurs and b does not occur (i.e. the non–action $\overline{b}$ takes place), or
2) the action b is executed and a does not occur (i.e. $\overline{a}$ takes place).

If we try to define a process P as a common behaviour of P_1 and P_2, then we discover that there exists no common possible behaviour, that each combination of requirements of P_1 and P_2 is *contradictory.* In the first case, it is required that, on the one hand, the actions a and b should occur, on the other hand, the action b can not occur (if the alternative action a is chosen to be performed). In the second case, the similar situation occurs concerning the occurrence of the action a. In such situations we will say that the action b (or, correspondingly, the action a) is deadlocked.

To denote *the deadlocked actions* we use the alphabet

$$\Delta_\alpha = \{\delta_a, \delta_b, \delta_c, \ldots\}.$$

The process defined by the SAFP$_2$ formula will be characterized by a set of partial orders in the alphabet $\alpha \cup \overline{\alpha} \cup \Delta_\alpha$.

A partially ordered set (poset) is a pair $p = (V, <)$ consisting of

(i) a vertex set V, typically modeling process actions, non–actions and deadlocked actions, i.e. $V \subseteq \alpha \cup \overline{\alpha} \cup \Delta_\alpha$;
(ii) a partial order $<$ over V, with $a < b$ typically interpreted as the action a necessarily preceding the action b in process.

Let us denote by $V^+ = \{x \in V \mid x \in \alpha\}$ — *the action subset of* V,

$V^- = \{x \in V \mid x \in \overline{\alpha}\}$ — *non–action subset of* V,

and $\Delta_V = \{x \in V \mid x \in \Delta_\alpha\}$ — *deadlocked action subset of* V,

i.e. $V = V^+ \cup V^- \cup \Delta_V$.

Remark.
In fact, we will consider posets $(V, <)$ of two types only: either $V = V^+ \cup V^-$ or $V = V^+ \cup \Delta_V$, because if there is some deadlocked actions in the poset $(V, <)$ then we will not distinguish between non-actions and deadlocked actions any more, and announce all "actions" in the "negative" part as deadlocked ones.

In this section we will consider posets, which satisfy the following conditions:

1) action v, non–action $\overline{v}$ and deadlocked action δ_v do not occur in poset p together, i.e. the action v occurs in p (and exactly once), or non–actions v occurs in p, or deadlocked action δ_v takes place in p;
2) partial order relation $<$ over V is irreflexive:
3) $\forall\ x,\ y \in V^- \cup \Delta_V : (x \not< y) \& (y \not< x)$, i.e. all elements of V^- (subset of non–actions) and Δ_V (subset of deadlocked actions) are incomparable;
4) $\forall\, x \in V^+, \rceil \exists y \in V^- \cup \Delta_V : (x < y) \vee (y < x)$, i.e. the elements of action subset V^+ and elements of $V^- \cup \Delta_V$ are also incomparable.
5) if there exists some deadlocked action δ_a such that $\delta_a \in V$, then $V^- = \emptyset$, i.e. $V = V^+ \cup \Delta_V$.

Now, we give some informal comments concerning the conditions defined above.

In our approach, each action in a process formula has its own unique name. Three different situations (excluding each other) can arise during process functioning:

1) either the action a occurs (and exactly once)

   or the action a does not occur and we distinguish two different reasons for arising such situation:

2) the non–action $\overline{a}$ takes place if some alternative action to the action a occurs, or

3) the deadlocked action δ_a takes place if the action a can't occur as a result of some mistake in a process specification.

We consider a process behaviour as a partially ordered set of actions added by some information about the actions which do not occur in this concrete process behaviour but which have been specified in a process formula.

All of the non–actions and deadlocked actions are incomparable, because for our purposes it does not matter: in which "order" two actions do not occur.

Thus, poset $p = (V, <)$ consists of two parts:

1) "*real*" (main) part, containing the actions of V^+ and defining, in fact, process behaviour, and
2) "*imaginary*" one, containing either non–actions of V^- or deadlocked actions of Δ_V.

Moreover, $< \cap (V^+ \times V^+) =<$, and $< \cap (V^- \times V^-) = \emptyset$, and $< \cap (\Delta_V \times \Delta_V) = \emptyset$. Thus, $p = (V, <) = (V^+, <) \cup (V^-, \emptyset) \cup (\Delta_V, \emptyset)$.

To define the denotational semantics of the basic process operation we will introduce the similar *poset operations:* ; (precedence), $\|$ (concurrency), $\triangledown$ (alternative).

If the poset p, constructed by means of these operations does not satisfy the conditions 1)–5) mentioned above we "correct" it using new auxiliary *regularization operation* $[p]$.

This operation singles out the maximal prefix of p, satisfying the conditions 1)–5) "before" some contradictions in process specification arise. All the actions specified in this process behaviour occurring "after" these happened contradictions, are announced as the deadlocked actions.

At first, we distinguish the actions which are directly do not satisfy the requirements 1)–5) in such a new constructed poset $p = (V, <)$ and the actions which have been announced earlier (by similar reasons) as the deadlocked ones (i.e. $\Delta_V \subseteq V$). We will call these actions as the primary deadlocked actions:

$$D_1 = \{\delta_v \,|\, (v \in V) \,\&\, ((v, v) \in <)\} \cup \{\delta_v \,|\, (v \in V) \,\&\, (\overline{v} \in V)\} \cup \{\delta_v \,|\, (v \in V) \,\&\, (\delta_v \in V)\} \cup \{\delta_v \,|\, (\overline{v} \in V) \,\&\, (\delta_v \in V)\} \cup \Delta_V .$$

All of the actions included in the set D_1 can not occur as a result of contradictory requirements for their occurrence in a process functioning.

At the second step, we find all the actions which have been specified in a process to occur "after" the actions of D_1, and hence they also can not occur during process functioning.

So, they should be also announced as deadlocked actions:

$$D_2 = \{\delta_w \,|\, (w \in V) \,\&\, ((\delta_v,\, w) \in <) \,\&\, (\delta_v \in D_1)\}\,.$$

If $D_1 \cup D_2 \neq \emptyset$ then we will not distinguish non-actions and deadlocked actions in this process any more, because, in fact, there is no big difference between non-actions and deadlocked actions in this process any more. If even there is only one deadlocked action in the process P, nevertheless in the sequential composition $(P;Q)$ all the actions of Q will never occur and will be announced as deadlocked ones, and if we consider concurrent composition $(P\|Q)$ and there is some action a specified in Q to occur, then it does not matter by which reason in the process P the action a can't occur (i.e. if it is $\overline{a}$ or δ_a), the result will be the same: the action a becomes deadlocked one (i.e. δ_a) in the process $(P\|Q)$.
Thus, if $D_1 \cup D_2 \neq \emptyset$ then we add to the set of deadlocked actions the non-actions as well:

$$D = D_1 \cup D_2 \cup \{\delta_v | \overline{v} \in V\}$$

Let us now define formally the regularization operation [] on the wrong–constructed posets:

$$[p] = (D,\, \emptyset) \cup (V \backslash \widehat{\alpha}(D),\, < \cap ((V \backslash \widehat{\alpha}(D)) \times (V \backslash \widehat{\alpha}(D)))\,.$$

It easy to verify, that if the poset satisfies the conditions 1)–5) mentioned above, then $[p] = p$.

Now, we define *the poset operations* in the following way.

Precedence. The precedence operation $(p_1\,;\, p_2)$ of two posets is defined as:

$$p = (V_1, <_1)\,;\, (V_2, <_2) = [(V_1 \cup V_2, <_1 \cup <_2 \cup (V_1^+ \times V_2^+) \cup (\Delta_{V_1} \times V_2^+))],$$

i.e. in a new poset p each action of p_1 precedes each actions of p_2; or if the constructed object does not satisfy the 1)–5) conditions then we "correct" it using regularization operator.

Example. Let $p_1 = (\{a\}, \emptyset)$ and $p_2 = (\{b\},\, \emptyset)$, then

$$p_3 = (p_1\,;\, p_2) = (\{a,\, b\}, <_3)\,, \text{ where } a <_3 b\,.$$

But

$$p_4 = (p_3\,;\, p_2) = [(\{a,\, b\}, <_3 \cup \emptyset \cup (a,\, b) \cup (b,\, b))] = (\{a,\, \delta_b\}, \emptyset)\,,$$

because we have contradictory requirements for the occurrence of the action b, and as result, the action b is deadlocked.

Concurrency. The concurrency operation $(p_1 \parallel p_2)$ of two posets is defined as:

$$p = (V_1, <_1) \parallel (V_2, <_2) = [(V_1 \cup V_2, (<_1 \cup <_2)^*)],$$

where $(<_1 \cup <_2)^*$ is a transitive closure of relation $(<_1 \cup <_2)$.

Example. Let $p_1 = (\{a, c\}, <_1)$, where $a <_1 c$, and $p_2 = (\{b, c\}, <_2)$, where $b <_2 c$. Then $p_3 = (p_1 \parallel p_2) = (\{a, b, c\}, <_3)$, where $a <_3 c$ and $b <_3 c$.

Let us consider additionally: $p_4 = (\{a, c\}, <_4)$, where $c <_4 a$. Then

$$\begin{aligned} p_5 = (p_3 \parallel p_4) &= [(\{a, b, c\}, (a, c) \cup (b, c) \cup (c, a))^*)] \\ &= [(\{a, b, c\}, ((a, c) \cup (b, c) \cup (c, a) \cup (a, c) \cup (c, c))] \\ &= (\{b, \delta_a, \delta_c\}, \emptyset). \end{aligned}$$

To define the next operations we introduce two new notions.

Let $p_1 = (V_1, <_1)$ and $p_2 = (V_2, <_2)$ be the posets.

The poset p_1 is a *prefix* of poset p_2 $(p_1 \subset p_2)$ iff

$$V_1^+ \subset V_2^+, <_2 \cap (V_1^+ \times V_1^+) = <_1,$$

and

$$\forall x \in V_1^+, \forall y \in V_2^+ : y <_2 x \Longrightarrow y \in V_1^+.$$

The modified union $\tilde{\cup}$ is defined as follows:

$$\{p_1\} \,\tilde{\cup}\, \{p_2\} = \begin{cases} \{p_1\}, & \text{if} \quad p_2 \subseteq p_1 \\ \{p_2\}, & \text{if} \quad p_1 \subseteq p_2 \\ \{\{p_1\}, \{p_2\}\}, & \text{otherwise.} \end{cases}$$

In the last case, when p_1 and p_2 are not prefixes of each other and they are not equal we have a set consisting of the posets p_1 and p_2.

The modified union for posets absorbs the computations, which can be continued by another one. Some examples will be given later.

Alternative. The alternative $(p_1 \triangledown p_2)$ of two posets is define as follows:

$$p = (V_1, <_1) \triangledown (V_2, <_2) = \{[(V_1 \cup \overline{V}_2, <_1)]\} \,\tilde{\cup}\, \{[(\overline{V}_1 \cup V_2, <_2)]\},$$

where

$$\overline{V} = \{\overline{a} \mid a \in V^+\} \cup \{\overline{a} \mid \overline{a} \in V^-\} \cup \{\overline{a} \mid \delta_a \in \Delta_V\}.$$

It should be noted that $(p_1 \triangledown p_2)$ is not a partial order, but a set of two partial orders, describing possible alternative computations (process behaviours), namely: if p_1 occurs, then the actions of p_2 do not occur, and vice versa.

Example. Let $p_1 = (\{a\}, \emptyset)$, $p_2 = (\{b\}, \emptyset)$, then

$$(p_1 \triangledown p_2) = \{(\{a, \overline{b}\}, \emptyset) \cup \{(\{\overline{a}, b\}, \emptyset)\}.$$

We extend the operation introduced above for sets of partial orders in the natural way.

Let $P_1 = \overset{n}{\underset{i=1}{\cup}} \{p_1^i\}$ and $P_2 = \overset{k}{\underset{j=1}{\cup}} \{p_2^j\}$ be the sets of partial orders, then

$$P_1 \circ P_2 = \overset{n}{\underset{i=1}{\tilde{\cup}}} \left(\overset{k}{\underset{j=1}{\tilde{\cup}}} \{p_1^i \circ p_2^j\} \right),$$

where $\circ \in \{;, \|, \triangledown\}$.

Nondeterministic concurrent process is characterized by the set of partial orders, associated with all possible (alternative) process behaviours.

Let us denote by $\mathcal{D}_2[\![P]\!]$ *the set of partial orders* associated with a process P.

Denotational semantics of SAFP_2 formulae is defined as follows:

1) $\mathcal{D}_2[\![a]\!] = (\{a\}, \emptyset)$.
2) $\mathcal{D}_2[\![P \| Q]\!] = \mathcal{D}_2[\![P]\!] \| \mathcal{D}_2[\![Q]\!]$
3) $\mathcal{D}_2[\![P; Q]\!] = \mathcal{D}_2[\![P]\!]; \mathcal{D}_2[\![Q]\!]$
4) $\mathcal{D}_2[\![P \triangledown Q]\!] = \mathcal{D}_2[\![P]\!] \triangledown \mathcal{D}_2[\![Q]\!]$

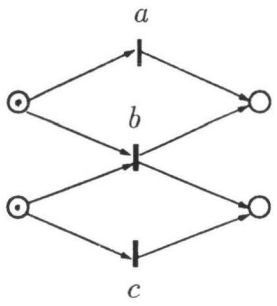
Now, we can explain the necessity and meaning of the modified union $\tilde{\cup}$ defined above.

Example. Let us consider the following process specification:

$$P = (a \triangledown b) \,\|\, (b \triangledown c).$$

It consists of two concurrently combining subprocesses: $P_1 = (a \triangledown b)$ and $P_2 = (b \triangledown c)$.

If we try to imagine the corresponding net representation for the process P then it will be the following:



The process P_1 specified by the formula $(a \triangledown b)$ consists of two different behaviours: if the action a occurs, then the action b does not occur and vice versa. The process P_2 has a similar semantics. Let us denote:

$$p_1^1 = (\{a, \overline{b}\}, \emptyset),\ p_1^2 = (\{b, \overline{a}\}, \emptyset),$$

and

$$p_2^1 = (\{b, \overline{c}\}, \emptyset),\ p_2^2 = (\{c, \overline{b}\}, \emptyset).$$

In our approach, we suppose that each action has its own unique name. So, in a process $P = (a \triangledown b) \parallel (b \triangledown c)$, the action b can occur iff it occurs in both subprocesses: $P_1 = (a \triangledown b)$ and $P_2 = (b \triangledown c)$. Thus, the behaviour p_1^1 of the subprocess P_1 takes place (occurs) in the compound process P iff the behaviour p_2^2 of the subprocess P_2 occurs. Correspondingly, the behaviour p_1^2 of the subprocess P_1 occurs in the composite process P iff the behaviour p_2^1 of the subprocess P_2 occurs.

However, if we define the semantics of the process formula $P = (a \triangledown b) \parallel (b \triangledown c)$ by the formal rules, we should combine each possible process behaviour of $P_1 = (a \triangledown b)$ (namely: p_1^1, p_1^2) with each possible process behaviour of $P_2 = (b \triangledown c)$ (namely: p_2^1, p_2^2), even if their combination can not be chosen during the composite process functioning.

Thus,

$$\begin{aligned}\mathcal{D}_2[\![P]\!] = \mathcal{D}_2[\![P_1]\!] \parallel \mathcal{D}_2[\![P_2]\!] = \ & (p_1^1 \parallel p_2^1) \mathbin{\tilde{\cup}} (p_1^1 \parallel p_2^2) \mathbin{\tilde{\cup}} (p_1^2 \parallel p_2^1) \mathbin{\tilde{\cup}} (p_1^2 \parallel p_2^2) \\ & = [(\{a, \overline{b}, b, \overline{c}\}, \emptyset)] \mathbin{\tilde{\cup}} (\{b, \overline{a}, \overline{c}\}, \emptyset) \mathbin{\tilde{\cup}} \\ & (\{a, c, \overline{b}\}, \emptyset) \mathbin{\tilde{\cup}} [(\{b, c, \overline{a}, \overline{b}\}, \emptyset)].\end{aligned}$$

The first and fourth posets do not satisfy condition 1), which we claim for posets, underlying in our semantical domain, and by this reason, they should be regularized. At the next step, we obtain the following four posets:

$$\{(\{a, \delta_c, \delta_b\}, \emptyset)\} \mathbin{\tilde{\cup}} \{(\{b, \overline{a}, \overline{c}\}, \emptyset)\} \mathbin{\tilde{\cup}} \{(\{a, c, \overline{b}\}, \emptyset)\} \mathbin{\tilde{\cup}} \{(\{c, \delta_a, \delta_b\}, \emptyset)\}.$$

It is easy to note, that in spite of the fact that the first and fourth posets contain deadlocked actions, they do not reflect some mistake in the initial process specification. They were obtained as a result of syntactically "wrong" (practically impossible) combining some processes behaviours of P_1 and P_2. In such a case, there exist the other "correct" process behaviours which "continue" these syntactically "wrong" ones.

To eliminate such "wrong–constructed" behaviours we use the operation of modified union. The modified union $\widetilde{\cup}$ for posets absorbs the posets which are prefixes of some another one.

In our example, the posets $(\{a, \delta_c, \delta_b\}, \emptyset)$ and $(\{c, \delta_a, \delta_b\}, \emptyset)$ are prefixes of the poset $(\{a, c, \overline{b}\}, \emptyset)$.

Therefore:

$$\begin{aligned} & \{(\{a, \delta_c, \delta_b\}, \emptyset)\} \,\widetilde{\cup}\, \{(\{b, \overline{a}, \overline{c}\}, \emptyset)\} \,\widetilde{\cup} \\ & \{(\{a, c, \overline{b}\}, \emptyset)\} \,\widetilde{\cup}\, \{(\{c, \delta_a, \delta_b\}, \emptyset)\} \\ = \; & \{(\{b, \overline{a}, \overline{c}\}, \emptyset)\} \cup \{(\{a, c, \overline{b}\}, \emptyset)\}. \end{aligned}$$

Thus,

$$\mathcal{D}_2[\![p]\!] = \{(\{b, \overline{a}, \overline{c}\}, \emptyset), (\{a, c, \overline{b}\}, \emptyset)\}.$$

It should be noted, that if some process behaviour contains "*real*" *deadlock*, then there is no another process behaviour which continues this one.

2.3 Full abstractness

One of the purposes of a concurrency theory is to analyze and verify statements about processes, in particular, to determine whether two processes specified by different algebraic formulas can be identified.

One of the natural ways of reasoning is to identify two processes if they generate the same sets of possible behaviours. Each behaviour in such a set is a result of some nondeterministic choice among the alternative actions during a run of the initial process. Thus, really, each process behaviour can be considered as a poset consisting of the process actions.

So, if $p = (V, <)$ is a poset and $V \subseteq \alpha \cup \overline{\alpha} \cup \Delta_\alpha$, then let $p^+ = (V^+, <)$ denote its *"real", "observable", part* over the action subset.

Correspondingly, if process P is characterized by the set of partial orders $\{p_i\}_{i=1}^n$ (i.e. $\mathcal{D}_2\ [\![P]\!] = \{p_i\}_{i=1}^n$) then let $\mathcal{D}_2^+\ [\![P]\!] = \{p_i^+\}_{i=1}^n$ denote its *"observable" part* over the action subset.

Two processes P and Q are *observationally equivalent*, written $P \approx_+ Q$, iff

$\mathcal{D}_2^+ [\![P]\!] = \mathcal{D}_2^+ [\![Q]\!]$.

By this definition, the following two processes $P = (a \triangledown b)$ and $Q = (a \| b) \| (a \triangledown b)$ are observationally equivalent (because $\mathcal{D}_2[\![P]\!] = \{(\{a, \overline{b}\}, \emptyset), (\{b, \overline{a}\}, \emptyset)\}$ and $\mathcal{D}_2[\![Q]\!] = \{(\{a, \delta_b\}, \emptyset), (\{b, \delta_a\}, \emptyset)\}$, hence $\mathcal{D}_2^+[\![P]\!] = \mathcal{D}_2^+[\![Q]\!]$ and can not be distinguished by any observations of partially ordered actions).

However, if we consider processes $(P; c)$ and $(Q; c)$ then we obtain behaviourally different processes:

$\mathcal{D}_2[\![(a \triangledown b); c]\!] = \{(\{a, c, \overline{b}, <_1), (\{b, c, \overline{a}\}, <_2)\}$, where $a <_1 c$ and $b <_2 c$.
$\mathcal{D}_2[\![((a \| b) \| (a \triangledown b)); c]\!] = \{(\{a, \delta_b, \delta_c\}, \emptyset), (\{b, \delta_a, \delta_c\}, \emptyset)\}$.
$\mathcal{D}_2^+[\![P; c]\!] \neq \mathcal{D}_2^+[\![Q; c]\!]$

Thus, observationally equivalent processes can become non–equivalent if they are placed in the same process context (or process environment), i.e. $\approx_+$ is not a congruence.

So, if we would like to consider the equivalent processes as modules that can be mutually exchanged in any context without affecting the observable behaviour of the latter, we need **both** a *"positive", "observable"* information about the actions which a process can perform (during its possible behaviour) and a *"negative", "unobservable"* information concerning the actions which can not occur in this process behaviour.

A *context* $\mathcal{C}[\,]$ is an expression with zero or more "holes" to be filled by an expressions. We write $\mathcal{C}[P]$ for the result of placing P in each "hole".

Lemma 2.1
Let P and Q be a processes from SAFP_2.

If $\forall \mathcal{C}[\,] : \ \mathcal{C}[P] \approx_+ \mathcal{C}[Q]$ then $\alpha(P) = \alpha(Q)$,

i.e. if the processes P and Q are observationally equivalent for any context $\mathcal{C}[\,]$, then they have the same action alphabet.

Proof.
Let us suppose the contrary, that $\alpha(P) \neq \alpha(Q)$, i.e. there exists such an action a that $a \in \alpha(P)$ but $a \notin \alpha(Q)$.

If $a \in \alpha(P)$ then for any partial order $p_i = (V_i^P <_i^P)$ such that $p_i \in \mathcal{D}_2[\![P]\!]$ the following assertion is valid:

$$(a \in V_i^P) \ \underline{\text{or}} \ (\overline{a} \in V_i^P) \ \underline{\text{or}} \ (\delta_a \in V_i^P)$$

However, $P \approx_+ Q$. It means that, at least, "positive" parts of processes P and Q are

the same. Since $a \in \alpha(P)$ and $a \notin \alpha(Q)$ we have $\forall p_i \in \mathcal{D}_2[\![P]\!]$:

$$(\overline{a} \in V_i^P) \underline{\text{ or }} (\delta_a \in V_i^P).$$

Let us consider the following processes: $(P\|a)$ and $(Q\|a)$. By the lemma condition it has to be: $(P\|a) \approx_+ (Q\|a)$.

However,

$$\forall p_i \in \mathcal{D}_2[\![P\|a]\!] : \delta_a \in V_i^{P\|a},$$

$$\forall q_i \in \mathcal{D}_2[\![Q\|a]\!] : a \in V_i^{Q\|a}.$$

Hence, $\mathcal{D}_2^+[\![P\|a]\!] \neq \mathcal{D}_2^+[\![Q\|a]\!]$ and $(P\|a) \not\approx_+ (Q\|a)$.

Thus, our initial proposition was wrong and $\alpha(P) = \alpha(Q)$. ■

Theorem 2.1
Denotational semantics $\mathcal{D}_2$ for processes of SAFP$_2$ is fully abstract w.r.t. $\approx_+$, i.e.

$$\forall \mathcal{C}[\,] : \quad \mathcal{C}[P] \approx_+ C[Q] \quad \Leftrightarrow \quad \mathcal{D}_2[\![P]\!] = \mathcal{D}_2[\![Q]\!]$$

Proof.
$\Rightarrow$
Let us suppose the contrary, that there are such processes P and Q, that $\forall \mathcal{C}[\,]$: $\mathcal{C}[P] \approx_+ \mathcal{C}[Q]$ but $\mathcal{D}_2[\![P]\!] \neq \mathcal{D}_2[\![Q]\!]$.

Using Lemma 2.1, we have that $\alpha(P) = \alpha(Q)$, and also by theorem condition: $P \approx_+ Q$. It means that at least "positive" parts of P and Q are the same: $\mathcal{D}_2^+[\![P]\!] = \mathcal{D}_2^+[\![Q]\!]$. But we assumed that $\mathcal{D}_2[\![P]\!] \neq \mathcal{D}_2[\![Q]\!]$, hence the possible "difference" has to be in the "negative" parts of P and Q. It means, that there exist such $p_{i_o} \in \mathcal{D}_2[\![P]\!]$ (where $p_{i_o} = (V_{i_o}^P <_{i_o}^P)$) and $q_{j_o} \in \mathcal{D}_2[\![Q]\!]$ (where $q_{j_o} = (V_{j_o}^Q, <_{j_o}^Q)$) that $p_{i_o}^+ = q_{j_o}^+$, but their "negative" parts are different.

As it has been noticed earlier each partial order considered as a semantics for SAFP$_2$ processes has negative part consisting of either non-actions only or deadlocked actions only,

i.e. either $V_{i_o}^P = (V_{i_o}^P)^+ \cup (V_{i_o}^P)^-$ and $\Delta_{V_{i_o}}^P = \emptyset$ in this case,
or $V_{i_o}^P = (V_{i_o}^P)^+ \cup (\Delta_{i_o}^P)$ and $(V_{i_o}^P)^- = \emptyset$ in this case.

Exactly the same is true for the vertex set of $q_{j_o} = (V_{j_o}^Q, <_{j_o}^Q)$. If p_{i_o} and q_{i_o} are different in their negative parts, then we can distinguish two cases:

(i) if $(V_{i_o}^P)^- \neq \emptyset$ then $\Delta_{V_{j_o}}^Q \neq \emptyset$

(ii) if $\Delta_{V_{i_o}}^P \neq \emptyset$ then $(V_{j_o}^Q)^- \neq \emptyset$

And it also has to be mentioned that there are no others different partial orders $p_i \in \mathcal{D}_2[\![P]\!]$ and $q_j \in \mathcal{D}_2[\![Q]\!]$ which have the same "positive" parts as p_{i_o} and q_{i_o} (by definition of $\tilde{\cup}$, which absorbs all prefixes).

Now, let us consider the processes $(P;c)$ and $(Q;c)$, where $c \notin \alpha P \cup \alpha Q$.

$$\mathcal{D}_2[\![P;c]\!] = \{(p_i;(\{c\},\emptyset))|p_i \in \mathcal{D}_2[\![P]\!]\},$$
$$\mathcal{D}_2[\![Q;c]\!] = \{(q_i;(\{c\},\emptyset))|q_i \in \mathcal{D}_2[\![P]\!]\}.$$

If the situation (i) has place, i.e.

$$p_{i_o} = (V_i^P, <_i^P) = ((V_i^P)^+, <_i^P) \cup ((V_i^P)^-, \emptyset)$$
$$q_{i_o} = (V_j^Q, <_j^Q) = ((V_j^Q)^+, <_j^Q) \cup ((\Delta_j^Q)^-, \emptyset),$$

then

$$p_{i_o};(\{c\},\emptyset)) = (V_{i_o}^P \cup \{c\}, <_{i_o}^P \cup (V_{i_o}^P)^+ \times \{c\}),$$
$$q_{j_o};(\{c\},\emptyset)) = (V_{j_o}^Q \cup \{\delta_c\}, <_{j_o}^Q).$$

So, $\mathcal{D}_2^+[\![P;c]\!] \neq \mathcal{D}_2^+[\![Q;c]\!]$ and $(P;c) \not\approx_+ (Q;c)$

The consideration of the case (ii) is similar.

It means, that with the conjecture that $\mathcal{D}_2[\![P]\!] \neq \mathcal{D}_2[\![Q]\!]$ we found such a context $\mathcal{C}[\,]$ for the processes P and Q that we can distinguish them, i.e. that $\mathcal{C}[P] \not\approx_+ \mathcal{C}[Q]$. But it is contradiction with the theorem condition.

Hence our conjecture was wrong and $\mathcal{D}_2[\![P]\!] = \mathcal{D}_2[\![Q]\!]$.

$\Leftarrow$
It is sufficient to prove that if $\mathcal{D}_2[\![P]\!] = \mathcal{D}_2[\![Q]\!]$ then for any SAFP$_2$ process R the following statements are valid;

$$\mathcal{D}_2[\![P;R]\!] = \mathcal{D}_2[\![Q;R]\!],$$
$$\mathcal{D}_2[\![P\|R]\!] = \mathcal{D}_2[\![Q\|R]\!],$$
$$\mathcal{D}_2[\![P \triangledown R]\!] = \mathcal{D}_2[\![Q \triangledown R]\!],$$

However, it is immediately follows from the definition of the semantics $\mathcal{D}_2$ for SAFP$_2$ processes:

$$\mathcal{D}_2[\![P \circ R]\!] = \mathcal{D}_2[\![P]\!] \circ \mathcal{D}_2[\![R]\!] = \mathcal{D}_2[\![Q]\!] \circ \mathcal{D}_2[\![R]\!] = \mathcal{D}_2[\![Q \circ R]\!], \text{ where } \circ \in \{;, \|, \nabla\}.\blacksquare$$

3 Net algebra AFP_o

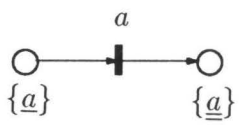
To answer the question: What class of nets could be specified by formulae of $SAFP_2$ we introduce the algebra AFP_o for the description of finite net structures.

3.1 The descriptive algebra AFP_O for building finite nets

Three basic operations of the algebra $SAFP_2$, namely: $\|$–concurrency, ;–precedence and ∇–alternative, can also be used as a basis for a net constructor. We will denote this *descriptive algebra* by AFP_o.

The proposed algebra AFP_o is generated by the class of atomic nets using the set of the net operations.

An atomic net is a net of the following form:



where a is a transition symbol, $\{\underline{a}\}$ is a head net place, $\{\underline{\underline{a}}\}$ is its tail place.

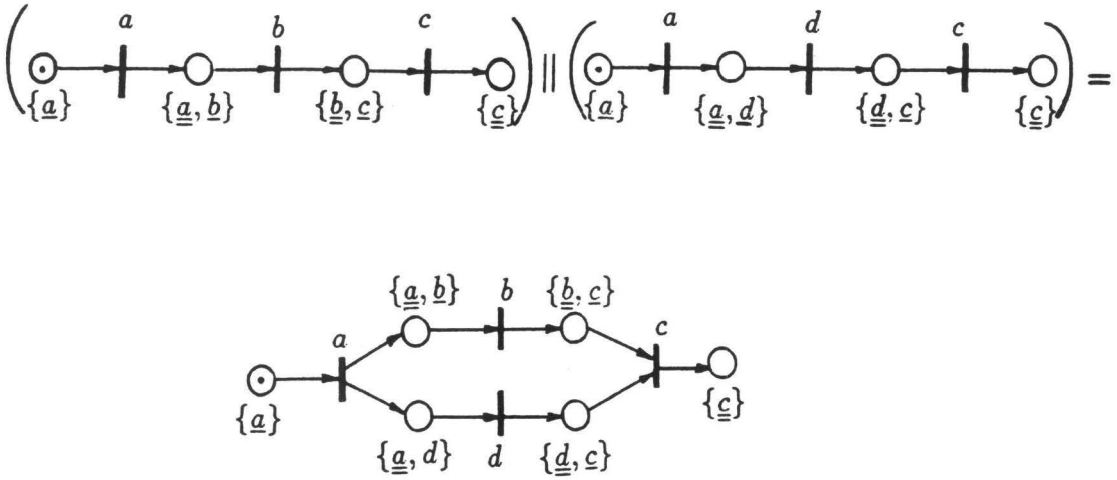
The concurrency operation (denoted"$\|$") is defined as a common graph union: it superposes one net on another (see Fig. 1a).

If $N_1 = (P_1,\ T_1\ F_1)$ and $N_2 = (P_2,\ T_2,\ F_2)$ then

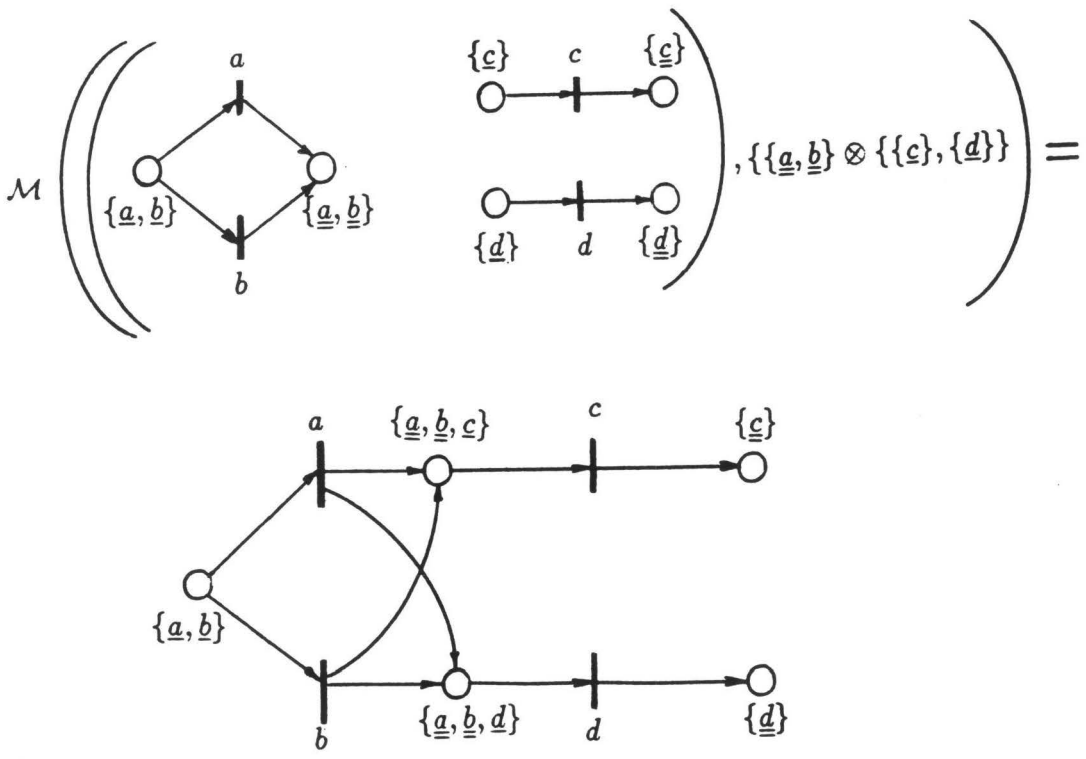
$$N = (N_1 \,\|\, N_2) = (P_1 \cup P_2,\ T_2 \cup T_2,\ F_1 \cup F_2)$$

Let $H(N)$ denote the set of *head places* of a net N and $G(N)$ be the set of *tail places* of N. By definition, $H(N) = H(N_1) \cup H(N_2)$ and $G(N) = G(N_1) \cup G(N_2)$.

Remark 1.
Here and further we will assume that any constructed net has a token in its each head place.

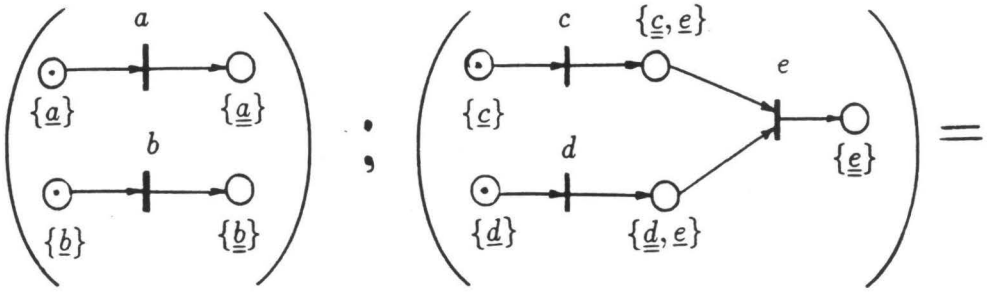


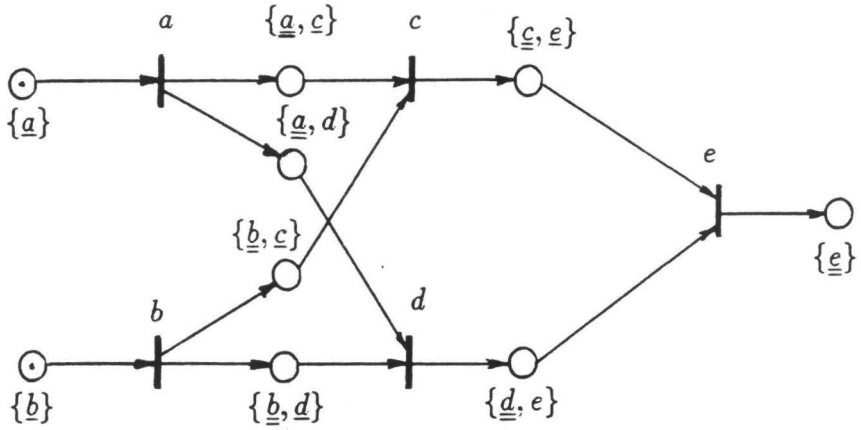
a) Concurrency



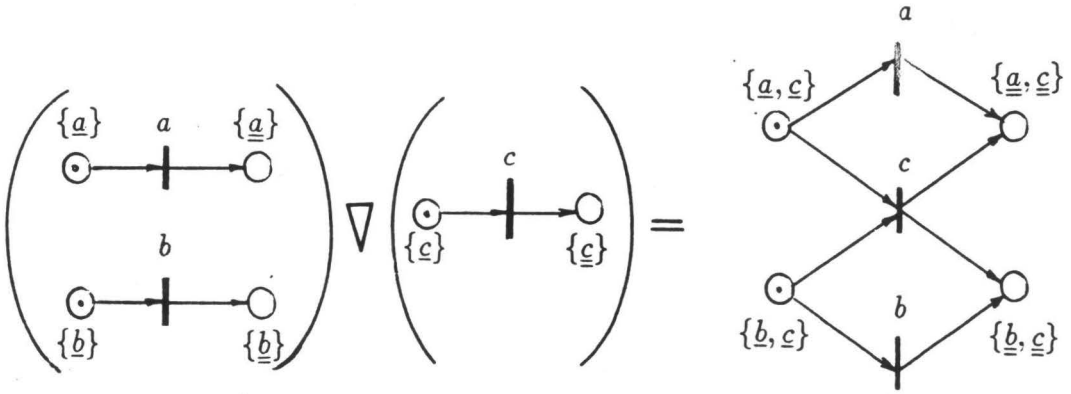
b) Merging

Figure 1: Net Operations





c) Precedence



d) Alternative

Figure 1: Net Operations (continued)

Other net operations can be defined via concurrency operation and the auxiliary merging operation. The latter merges two sets of places in a specific way. This involves two suboperations: 1) the formation of the set of merged places, 2) the replacement of the two existing sets by a new set.

Given two sets of places X and Y *the forming operation* "$\otimes$" results in the set Z of merged places:

$$Z = X \otimes Y = \{x \cup y \,|\, x \in X,\ y \in Y\}.$$

The merging operation $\mathcal{M}$ merges two sets of places, X and Y, in a net $N = (P,\ T,\ F)$ and generates a new net

$$N' = \mathcal{M}(N,\ X \otimes Y) = (P',\ T',\ F'),$$

where

$$P' = P \backslash (X \cup Y) \cup (X \otimes Y),\ T' = T,$$
$$\forall\, p (p \in P' \& \ p \notin X \otimes Y) : F'(p) = F(p),$$
$$\forall\, p \in X \otimes Y : F'(p) = F(x) \cup F(y), \text{ where } p = x \cup y,\ x \in X,\ y \in Y.$$

An example of the merging operation is shown in Fig. 1b.

Remark 2.
To define below the operations of precedence ; and alternative $\triangledown$, we assume that formulae $(A\,;\ B)$ and $(A \triangledown B)$ do not contain the same symbols in both A and B. In the general case, if formulae A and B have the common symbols, we rename these symbols (for example, in B) and, at first, construct the net $(A\,;\ B')$ or $(A \triangledown B')$, where B' is a result of renaming B in such a way, that A and B' do not contain the same symbols. After that, in a new constructed net, the transitions, which have had the same names in the initial nets A and B, are identified (are superposed).

The precedence operation ";" joins two nets by merging the set of tail places of the first net with the set of head places of the second (see Fig. 1c).

$$N = (N_1\,;\ N_2) = \mathcal{M}\,((N_1 \,\|\, N_2),\ G(N_1) \otimes H(N_2)).$$

By definition,

$$H(N) = H(N_1),\ G(N) = G(N_2).$$

The alternative operation "$\triangledown$" unites two nets by merging their sets of head places and separately their sets of tail places (see Fig. 1d).

$$N = (N_1 \triangledown N_2) = \mathcal{M}(N',\ G(N_1) \otimes G(N_2))\,,$$

where

$$N' = \mathcal{M}((N_1 \parallel N_2),\ H(N_1) \otimes H(N_2))\,.$$

By definition,

$$H(N) = H(N_1) \otimes H(N_2),\ G(N) = G(N_1) \otimes G(N_2)\,.$$

Let α be a class of atomic nets, i.e. a class of transition symbols.

A net formula in the algebra AFP_o over basis α is defined as follows:

1) each symbols of α is a formula;
2) if A and B are formulae, then $(A \parallel B)$, $(A\,;\,B)$ and $(A \triangledown B)$ are formulae.

Remark 3.
To be more precise we should distinguish the denotation of SAFP_2 and AFP_o operation. If it is necessary, we will write: $\|_2$, $;_2$, $\triangledown_2$ for operations of SAFP_2 and $\|_o$, $;_o$, $\triangledown_o$ for operations of AFP_o.

It is easy to note, that the nets described by AFP_o formulae form a subclass of elementary net systems [Th87]. The nets of this subclass generate only finite behaviours.

3.2 Occurrence nets as the net-processes

The net behaviour can be characterized by the set of occurrence nets [Pet77], representing concurrent processes.

Occurrence nets (or O-nets) are acyclic nets with a unique token in each of their head places, which additionally satisfy the following restriction:

$$\forall p \in P > (|{}^{\cdot}p| \leq 1 \wedge |p^{\cdot}| \leq 1),$$

i.e. every internal net place has only one input and output transition.

Let us consider a net $N = (P, T, F)$ specified by AFP_o formula.

A net $N' = (P', T', F')$ is called a *subnet* of the net $N = (P, T, F)(N' \subseteq N)$
iff

$$P' \subseteq P, T' \subseteq T, F' \subseteq F \cap (P' \times T' \cup T' \times P')$$

The net N' is called *an O-subnet* of the net N, if

1) N' is a subnet of N,

2) N' is O-net,

3) $\forall t \in T' : \{p \in P | pFt\} \subseteq P'$ and $\forall p \in P' : F'(p,t) = F(p,t)$,

i.e. transition t in the O-subnet N' has the same set of input places and the same set of arcs connecting t with its input places as in the net N.

We will call an O-subnet N' of the net N *maximal* iff

1) there is no O-subnet N'' of the net N such that $N' \subseteq N''$ and $N' \neq N''$;

2) all the head places of N are head places of N', i.e. $H(N') = H(N)$.

Remark.
The maximal O-subnet $N' = (P', T', F')$ may have in the structure some absolutely isolated places (i.e. places $\{p | \ p^{\cdot} = {}^{\cdot}p = \emptyset\}$), because of the requirement 2) in the definition of maximal O-subnet.

If the tail places of the maximal O-subnet N' are not among the tail places of the initial net N then we supply (label) them by special failure symbol δ, stressing by this the unsuccessful (failure) termination of the net N functioning.

The set of all maximal O-subnets of net N forms the set of all possible concurrent processes generated by this net.

Examples.

1) Let us consider net $N_1 = (a \triangledown b) \| (b \triangledown c)$ in Fig. 2a.

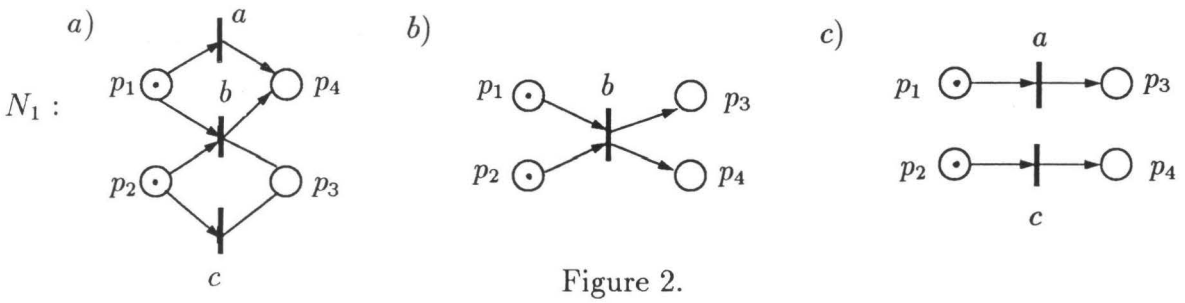


Figure 2.

The set of its maximal subnets is shown in Fig. 2b,c

2) Let us consider the net $N_2 = (a\triangledown(b;e))\|(d\triangledown(c;e))$ shown in Fig. 3.

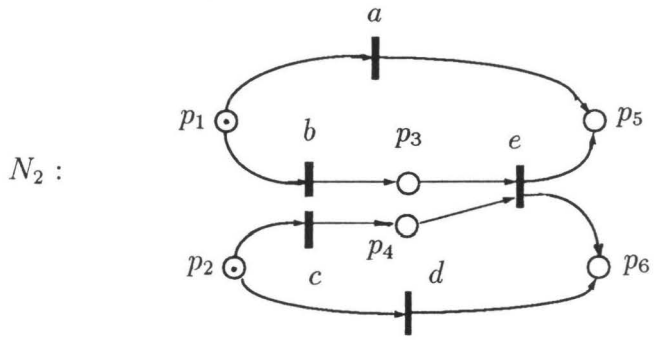


Figure 3.

Its set of maximal O-subnets is shown in Fig. 4a, b, c, d.

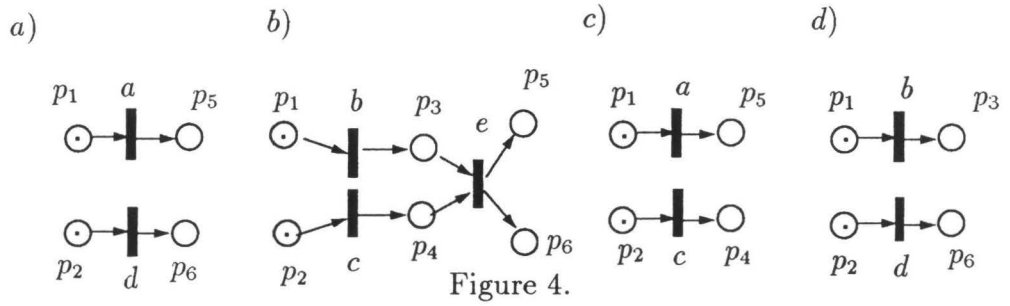


Figure 4.

3) For the net $N = a\|(a\triangledown b)\|b$ shown in the Fig. 5a the set of its maximal O-subnets is represented in the Fig. 5b,c.

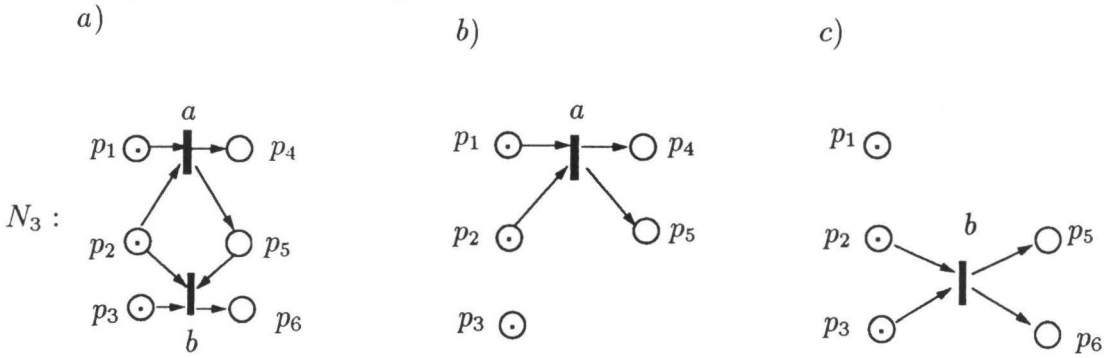


Figure 5.

Thus, in occurrence nets, we have, sometimes, isolated places, which have neither input transitions, nor output transitions.

3.3 Semantics for AFP$_o$ nets: algebraic approach

Since we consider the nets defined by the AFP$_o$–formulae, the set of concurrent processes (i.e. the set of maximal O–subnets) can be also defined by means of AFP$_o$ operations.

Let us denote by $\mathcal{D}_o[\![N]\!]$ the set of concurrent processes (i.e. the set of maximal O–subnets) associated with the defined by AFP$_o$ formula N.

The semantics $\mathcal{D}_o[\![N]\!]$ is defined as follows:

1. $\mathcal{D}_o[\![a]\!] = a$,
   i.e. the set of maximal O–subnets associated with the net defined by the AFP$_o$ formula a coincides with the initial A–net a.

   To define a semantics of next operations, we introduce the following notions.

   Let $N_1 = (P_1,\ T_1,\ F_1)$ be O–net. The O–net $N_2 = (P_2,\ T_2,\ F_2)$ is called a *prefix* of $N_1 (N_2 = pref\ (N_1))$ iff

   1) $P_2 \subseteq P_1,\ T_2 \subseteq T_1,\ F_2 \subseteq F_1$ and $F_1 \cap (P_2 \times T_2 \cup T_2 \times P_2) = F_2$
   2) $\forall\, x \in P_2 \cup T_2\ \forall\, y \in P_1 \cup T_1 : (y,x) \in F_1 \Longrightarrow y \in P_2 \cup T_2$ and $(y,x) \in F_2$.
   3) $H(N_1) = H(N_2)$, i.e. the sets of the head places $\{p|\cdot p = \emptyset\}$ for N, and N_2 are the same.

   In addition, if N_2 is a proper prefix of N_1 (i.e. $N_1 \neq N_2$) then the tail places of N_2 i.e. the places $\{p|p^\cdot = \emptyset\}$ are supplied by special label δ specifying failure (noncomplete) termination of initial O–net N_1.

   Let $N_1, \ldots, N_n$ be the O–nets.

   Then

$$\tilde{\bigcup_{i=1}^{n}} \{N_i\} = \{N_j\}_{i=1}^{n} \,|\, \forall\, j_1, j_2 (1 \leq j_1 \neq j_2 \leq n) : N_{j_1} \not\subseteq N_{j_1} \text{ and } N_{j_2} \not\subseteq N_{j_1}\}\,.$$

   i.e. *the modified union* $\tilde{\cup}$ absorbs among the set of O–nets such O–nets which are prefixes of the others ones.

   We will call two nets $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$) are *coherent* iff

$$\forall a, b \in T_1 \cap T_2 : a F_1^+ b \Leftrightarrow a F_2^+ b$$

We will call two nets $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$) are *coherent* iff

$$\forall a, b \in T_1 \cap T_2 : aF_1^+b \Leftrightarrow aF_2^+b$$

2. Let T_{AB} be a set of transition symbols which is common for the formulae A and B (i.e. $T_{AB} = \alpha(A) \cap \alpha(B)$). Then

$$\mathcal{D}_o[\![A \parallel B]\!] = \widetilde{\bigcup_{i,j}} \{< (A_i \parallel B_j) > \mid A_i \in \mathcal{D}_o[\![A]\!],\ B_j \in \mathcal{D}_o[\![B]\!]\},$$

where

$$< (A_i \parallel B_j) >= \{(A_i' \parallel B_j') \mid \begin{array}{l} A_i' = pref\ (A_i),\ B_j' = pref\ (B_j), \text{ and} \\ \alpha(A_i') \cap T_{AB} = \alpha(B_j') \cap T_{AB}, \text{ and} \\ A_i' \text{ and } B_j' \text{ are maximal coherent prefixes with such a} \\ \text{property } \}, \end{array}$$

i.e. composite process $(A \parallel B)$ consists of

(i) either completely independent subprocesses A_i and B_j (if $\alpha(A_i) \cap T_{AB} = \alpha(B_j) \cap T_{AB} = \emptyset$)
(ii) or correctly "synchronized" parts of the subprocesses A_i and B_j (if $\alpha(A_i) \cap T_{AB} \neq \emptyset$ and $\alpha(B_j) \cap T_{AB} \neq \emptyset$).

If $\alpha(A_i) \cap T_{AB} = \alpha(B_j) \cap T_{AB} \neq \emptyset$, then $< (A_i \parallel B_j) >= (A_i \parallel B_j)$ and it means that A_i and B_j can successfully terminate their executions, synchronizing executions of common transitions in A_i and B_j.

If $\alpha(A_i) \cap T_{AB} \neq \alpha(B_j) \cap T_{AB}$, it means that there exists a transition t, for example in O–net A_i, which have to be synchronized by its common execution in A_i and B_j (in according with requirement: $t \in T_{AB}$), but it is impossible, because $t \notin (\alpha(B_j) \cap T_{AB})$. Therefore, in this case to terminate execution of A_i successfully is also impossible, and only (noncontradictory) prefix of A_i can be executed.

3. $\mathcal{D}_o[\![A\,;\,B]\!] = \{< (A_i\,;\,B_j) > \mid A_i \in \mathcal{D}_o[\![A]\!],\ B_j \in \mathcal{D}_o[\![B]\!]\}$.

$$< (A_i\ ;\ B_j) >= \begin{cases} \{A_i\}, & \text{if the tail places of } A_i \text{ have a} \\ & \text{special label } \delta \text{ of failure termination} \\ (A_i\,;\,B_j), & \text{otherwise.} \end{cases}$$

i.e. composite process $(A\,;\,B)$ consists of

(i) either successfully terminated subprocess A_i after which B_j is executed
(ii) or only subprocess A_i if it has a special label of failure termination, which, in other words, means that some tokens stuck in internal places of A and, in this case, an execution of B is impossible.

4. $\mathcal{D}_o[\![A \triangledown B]\!] = \mathcal{D}_o[\![A]\!] \,\tilde{\cup}\, \mathcal{D}_o[\![B]\!]$,
   i.e. the set of maximal O–subnets associated with the net $(A \triangledown B)$ consists of the maximal O–subnets defined by the formulae A and B.

Remark.
As it has been defined earlier, $(A\,;\,B)$ and $(A \triangledown B)$ are AFP_o–formulae under the condition that formulae A and B do not contain the same transition symbols in both A and B (i.e. $\alpha(A) \cap \alpha(B) = \emptyset$).

By definition an O–subnet can contain several different places incident to the same set of transitions. For example, the O–subnet shown in Fig. 5b contains two different places p_1 and p_2 incident to the same transition a.

Let N be an O–subnet. Let us denote by $\tilde{N}$ a modified O–subnet constructed from N by means of the following rule:

$$\forall\, p_1,\ p_2 \in P\ :\ (p_1^{\cdot} = p_2^{\cdot} \wedge^{\cdot} p_1 =^{\cdot} p_2 \Longrightarrow p_1 = p_2),$$

i.e. the places incident to the same set of transitions are identified.

Lemma 3.1 [Ch89]

Let us consider A–net defined by AFP_o–formula N.

Let $\{N_i\}_{i=1}^{n}$ be the set of all maximal O–subnets of N. Then

$$\mathcal{D}_o[\![N]\!] = \{\ \tilde{N}_i \,|\, 1 \le i \le n\}\,.$$

3.4 Interrelation between algebras $\mathbf{SAFP_2}$ and $\mathbf{AFP_O}$

Now , we will establish that, in fact, it does not matter, if we consider the formula A as the AFP_o formula, or if we consider the same formula A as the SAFP_2 formula, it defines just the same process.

Let AFP_o and SAFP_2 be the algebras over the same action basis α.

Let us consider *the mapping* Ψ of the AFP_o formulae into the SAFP_2 formulae which is defined *in the following way*:

$$\begin{aligned}
&\Psi(a) = a, \text{ where } a \in \alpha,\\
&\Psi(A\,;_o\,B) = \Psi(A)\,;_2\,\Psi(B),\\
&\Psi(A \triangledown_o B) = \Psi(A) \triangledown_2 \Psi(B),\\
&\Psi(A \,\|_o\, B) = \Psi(A)\,\|_2\,\Psi(B).
\end{aligned}$$

As it has been noted earlier the net representation of a concurrent process practically coincides with its partial order representation. If N is an O–net then the associated partial order p_N can be constructed in the following way. Let T_N be a transition set of N, and $<_N$ be a basic order (precedence) relation (based on F_N) over T_N. Then $p_N = (T_N, <_N)$.

An O–net N are *po–equivalent* to partial order $p(N \approx_{po} p)$if the associated partial order p_n constructed by the O–net N coincides with p (i.e. $p = p_N$).

We can extend the notion of po–equivalence to the set of O–nets and partial orders in the natural way.

Let $p^+ = (V^+, <)$ denote the *restriction (projection)* of partial order $p = (V, <)$ on the action set α (i.e. $p^+ = (V^+, <)$ is a so–called "real" part or partial order $p = (V, <)$).

Let us consider a formula P of SAFP_2 and suppose that $\mathcal{D}_2[\![P]\!] = \bigcup_{i=1}^{n} \{p_i\}$.

Let also $\mathcal{D}_2^+[\![P]\!] = \bigcup_{i=1}^{n} \{p_i^+\}$.

Theorem 3.1 [Ch89]
Let us consider a Petri net described by the AFP_o formula N and let $\Psi(N) = P$ be the corresponding (just the same) SAFP_2 formula.

Then

$$\mathcal{D}_o[\![N]\!] \approx_{po} \mathcal{D}_2^+[\![P]\!].$$

We will say that the O-nets N, and N_2 are o-equivalent (occurrence-equivalent) and denote it: $N_1 \approx_o N_2$ iff

(i) $N_1 \approx_{po} N_2$, i.e. associated partial orders p_{N_1} and p_{N_2} constructed by O-nets N, and N_2 are the same: $p_{N_1} = p_{N_2}$

(ii) $\{p|p^{\cdot} = \emptyset$ and p has a special label δ of failure termination $\} \neq \emptyset$ for $N_1 \Leftrightarrow$
$\{p|p^{\cdot} = \emptyset$ and p has a special label δ of failure termination $\} \neq \emptyset$ for N_2,
i.e. if in the O-net N_1 has unsuccessful termination then the O-net N_2 terminates unsuccessfully as well, and vice versa.

We extend the notion of o-equivalence to the set of O-nets in natural way.

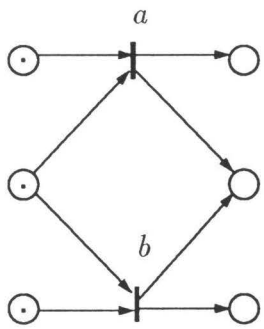
We will call two AFP_o nets N_1 and N_2 are *o-equivalent* ($N_1 \approx_o N_2$) iff their corresponding sets of maximal O-subnets are o-equivalent, i.e. $\mathcal{D}_o[\![N_1]\!] \approx_o \mathcal{D}_o[\![N_2]\!]$.

Examples

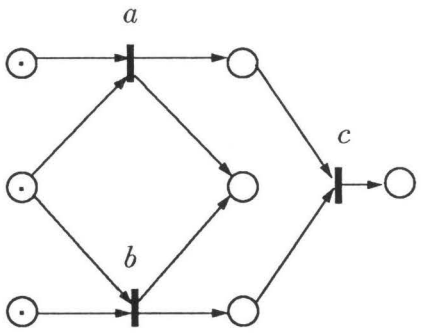
The following pairs of nets are o-equivalent.
We give also semantics based on posets with non-actions for the same process formulae to compare their "net" and "poset" representation.

1)

N_1 :



N_2 :



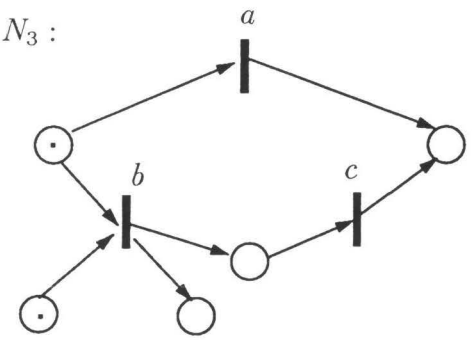
$P_1 = (a \| (a \triangledown b) \| b)$
$\mathcal{D}_2[\![P_1]\!] = \{(\{a, \delta_b\}, \emptyset), (\{b, \delta_a\}, \emptyset)\}$

$P_2 = ((a \| b); c) \| (a \triangledown b)$
$\mathcal{D}_2[\![P_2]\!] = \{(\{a, \delta_b, \delta_c\}, \emptyset), (\{b, \delta_a, \delta_c\}, \emptyset)$

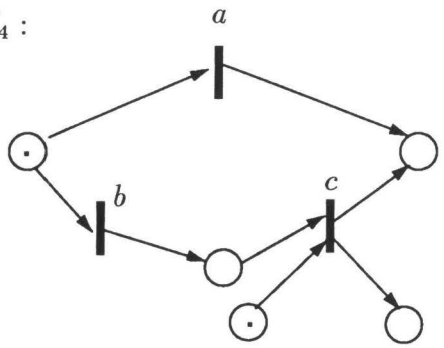
$$N_1 \approx_o N_2$$

2)

N_3 :



N_4 :



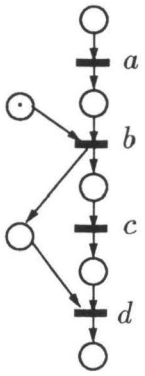
$P_3 = ((a \triangledown (b; c)) \| b)$
$\mathcal{D}_2[\![P_3]\!] = \{(\{a, \delta_b, \delta_c\}, \emptyset), (\{b, c, \overline{a}\}, <_3)\}$
where $b <_3 c$

$P_4 = ((a \triangledown (b; c)) \| c)$
$\mathcal{D}_2[\![P_4]\!] = \{(\{a, \delta_b, \delta_c\}, \emptyset), (\{b, c, \overline{a}\}, <_4)$
where $b <_4 c$

$$N_3 \approx_o N_4$$

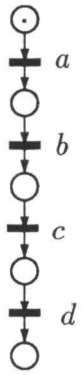
3)

N_5 :



$P_5 = (a; b; c; d) \| (b; d)$
$\mathcal{D}_2[\![P_5]\!] = (\{a, b, c, d\}, <_5)$
where $a <_5 b <_5 c <_5 d$

N_6 :

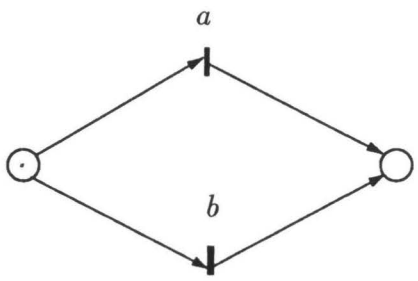


$P_6 = (a; b; c; d)$
$\mathcal{D}_2[\![P_6]\!] = (\{a, b, c, d\}, <_6)$
where $a <_6 b <_6 c <_6 d$

$$N_5 \approx_o N_6$$

Please, notice that the net N_1 is not o-equivalent to the net N_7 below. The nets N_1 and N_7 are po-equivalent (i.e. their sets of maximal O-subnets define the same set of partial orders on the actions, extracted from the occurrence nets), but maximal O-subnets of N_1 have the places labelled by δ (symbol of unsuccessful termination), in contrast of maximal O-subnets of N_7, terminated successfully.

N_7 :



$P_7 = (a \triangledown b)$
$\mathcal{D}_2[\![P_7]\!] = \{(\{a, \overline{b}\}, \emptyset), (\{b, \overline{a}\}, \emptyset)\}$

$N_1 \not\approx_o N_7$

Let us consider two Petri nets described by AFP_o formulae P_1 and P_2, such that $\alpha(P_1) = \alpha(P_2)$, i.e. the nets P_1 and P_2 are defined over the same sets of actions for

elementary nets, (since the function Ψ is identical we will not distinguish P_1 as formula of AFP_2 and P_2 as formula of SAFP_2).

The following theorem is a corollary of Theorem 3.1 and definition of net equivalence.

Theorem 3.2

$$\mathcal{D}_o[\![P_1]\!] \approx_o \mathcal{D}_o[\![P_2]\!] \text{ and } \alpha(P_1) = \alpha(P_2) \quad \Leftrightarrow \quad \mathcal{D}_2[\![P_1]\!] = \mathcal{D}_2[\![P_2]\!]$$

4 Operational semantics

4.1 Preliminary remarks

Denotational semantics for SAFP_2 considered above is based on the principle: to correspond for each operator *op* of SAFP_2 some its semantic equivalent $op^{\mathcal{D}}$ defined on the semantic domain and satisfying the following equation:

$$\mathcal{D}[\![op(P_1, P_2)]\!] = op^{\mathcal{D}}(\mathcal{D}[\![P_1]\!], \mathcal{D}[\![P_2]\!]).$$

Operational semantics is based on the principle: to define only the states and transitions of some abstract machine, functioning by similar rules as the described objects. The interleaving operational semantics of some process algebra takes as machine a nondeterministic automaton or labelled transition system [Ke76]. In our case, we replace the automaton by Petri net and the question is how to do this in the Plotkin's style of operational semantics [Pl81] where the distributed states and transitions of Petri nets have to be specified by some syntactic rules.

4.2 Different representation for subclass of Petri nets

In this section, for describing operational net semantics we will use the following (slightly different) definition for Petri nets [Cz85]:

Definition. A *Petri net* is a structure

$$N = (S, \rightarrow M_o),$$

where

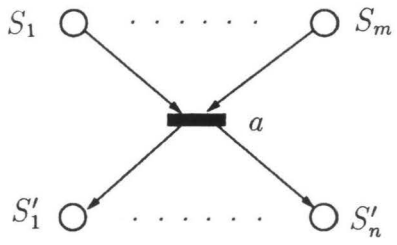
(i) S is a set of *places* or local *states*;

(ii) $\rightarrow \subseteq P_+(S) \times \alpha \times P_+(S)$ is the *transition relation*,

(iii) $M_o \subseteq P_+(S)$ is the *initial marking*.

Here, $P_+(S)$ denotes the set of non-empty subsets of a set S. An element $(P, a, Q) \in \rightarrow$ is called a (local) transition (labelled by a) and will be also written as $P \xrightarrow{a} Q$. For a transition $t = P \xrightarrow{a} Q$, its *preset* is defined by $pre(t) = P$, its *postsets* by $post(t) = Q$ and its *action* by $\alpha(t) = a$.

The graphical representation of Petri nets is as usual. Local states $s \in S$ are represented as circles with the name "S" outside and local transition $t = (\{S_1, \ldots, S_m\}, a, \{S'_1, \ldots S'_n\})$ as bars, labelled by "a" and connected via directed arcs to the local states in $pre(t)$ and $post(t)'$:



The initial marking M_o is represented by putting a token into the circle of each $s \in M_o$.

We consider here only acyclic nets. To this end we require that $pre(t)$ and $post(t)$ are disjoint.

The execution of the transition a transforms a marking M into a new marking M':

$$M \xrightarrow{a} M' \text{ iff } pre(a) \subseteq M \text{ and } M' = (M - pre(a)) \cup post(a).$$

A reachable marking of net N is a marking M such that:

$$M_o \xrightarrow{a_1} M_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} M_n = M$$

for some transitions $a_1, \ldots, a_n$ and markings $M_1, \ldots, M_n$, where M_o is the initial marking of N.

Let mark (N) denote the set of all reachable marking of N.

The reachable local states of N are defined by the set:

$$loc(N) = \{s \in S | \exists M \in \ mark(N) : \ s \in M\}$$

Further, we will identify the Petri nets which differ only by a one-one renaming of their reachable local states, i.e. we will identify the nets $N_i = (S_i, \rightarrow_i, M_i)$, $i = 1, 2$

$$N_1 \equiv N_2$$

if there exists a bijection $f : loc(N_1) \rightarrow loc(N_2)$ such that $\forall M \in mark(N_1)$ and all local transitions $P \xrightarrow{a}_1 Q$ of N_1 with $P \subseteq M$: $P \xrightarrow{a}_1 Q$ iff $f(P) \xrightarrow{a}_2 f(Q)$, and vice versa where $f(P)$ and $f(Q)$ are understood elementwise.

By an *abstract Petri net* we mean the equivalence class $N/_{\equiv}$ of some "concrete" nets N. The graphic representation for abstract net is the same as for concrete ones, except that places appear only for reachable local states and they have no any names $s \in S$.

We will characterize the semantics (behaviour) of the nets (abstract nets) by the set of pure concurrent processes generated by the net and specified by the set of maximal O-subnets (see definition in Section 3.2) and we will consider two nets N_1 and N_2 as equivalent (i.e. generating or describing the same behaviour) iff they are o-equivalent, i.e. if $N_1 \approx_o N_2$ (see definition in Section 3.4).

4.3 Decomposition of concurrent processes

Except the differences for the treatment of net equivalence and the transition rules for operations of SAFP$_2$, we use the same idea [DDM86, Ol88] of decomposing a process P into a set

$$\{\mathcal{P}, \ldots, \mathcal{P}_m\}$$

of sequential components which can be thought of as running concurrently. For example, we can split the concurrent process $(P \| Q)$ into the two sequential components-subprocesses P and Q (in the condition here, that P and Q themselves do not contain other concurrency operator $\|$ inside), but supplied, in the addition, with some information about their communication interface, i.e. the set of actions which have to be synchronized to be performed in P and Q : $A = \alpha P \cap \alpha Q$.
So, we will split $(P \| Q)$ into the

$$P_{A\|} \text{ and } Q_{\|A},$$

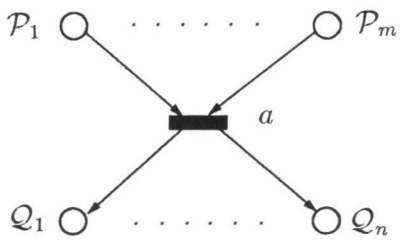
where $A = \alpha P \cap \alpha Q$.

Sequential components will denote the local states of Petri nets, and the local transitions will have a form

$$\{\mathcal{P}_1, \ldots, \mathcal{P}_m\} \xrightarrow{a} \{\mathcal{Q}_1, \ldots, \mathcal{Q}_n\},$$

where a is an action and $\mathcal{P}_1, \ldots, \mathcal{P}_m, \mathcal{Q}_1, \ldots, \mathcal{Q}_n$ are sequential components.

The graphical representation is following:



The set Seq of sequential components is defined by the following rules:

$$\mathcal{P} := \nu \mid a \mid (\mathcal{P})_{\mathcal{A}} | \mathcal{P}; Q | \mathcal{P} \triangledown \mathcal{Q}$$

Where ν is a special symbol of empty process, which does nothing and successfully terminates, a is an elementary action, i.e. $a \in \alpha$, and the index $\mathcal{A}$ is of the following type:

$$(*) \qquad \mathcal{A} := \emptyset | \ A \| \mid \| A \ | \mathcal{A}_2, \mathcal{A}_2, \qquad \text{where } A \subseteq \alpha$$

and describes some additional information about communication interfaces between the processes. We will call it communication information.

Please, notice that in the expression $(\mathcal{P}; Q)$ the symbol Q represents process of SAFP_2. However, we announce each process $(\mathcal{P}; Q)$ as a sequential component regardless whether Q contains any concurrency operator $\|$. Thus, sequentiality refers only to the first process in the sequential composition and require that it does not contain two actions that might occur concurrently.

Further, the following denotations will be used:

$a, b, c, \ldots$	for elementary actions from α
$A, B, C, \ldots$	for sets of actions from α
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$	for information about process communication interfaces of type (*)
$P, Q, R, \ldots$	for processes of SAFP_2
$\mathcal{P}, \mathcal{Q}, \mathcal{R}, \ldots$	for set of sequential components

Remark.
Before decomposing process to the set of sequential components, we are doing some additional syntactical analysis. If for processes $(P; Q)$ and $(P \triangledown Q)$ the following is valid: $\alpha(P) \cap \alpha(Q) \neq \emptyset$, then the actions from $\alpha(P) \cap \alpha(Q)$ can't occur, and such a process specification contains "syntactical" mistake. To recognize it and to show it explicitly, all the actions from $\alpha(P) \cap \alpha(Q)$ in the process formulae $(P; Q)$ and $(P \triangledown Q)$ are substituted by special deadlocked action symbol δ. The intuitive meaning of δ is that δ can't occur (be performed) in the process and cancel the performance of process actions specified to occur "after" it.

Decomposition of a process P into a set of sequential components is defined by the following *decomposition function* $dec(P)$:

1) $dec(a) = \{a\}$

2) $dec(P;Q) = dec(P);Q$

3) $dec(P\|Q) = (dec(P))_{A\|} \cup (dec(Q))_{\|A}$, where $A = \alpha(P) \cap \alpha(Q)$

4) $dec(P \triangledown Q) = dec(P) \triangledown dec(Q)$

(Remember, that if $\alpha(P) \cap \alpha(Q) = A \neq \emptyset$ in the processes $(P;Q)$ or $(P \triangledown Q)$, then all the action from A are substituted in subprocesses P and Q by δ).

In the clauses 2) - 4) the operations $;A\|$, $\|A$, and $\triangledown$ applied to the sets of sequential components are understood elementwise, i.e. for example
$(dec(P))_{A\|} = \{(\mathcal{P})_{A\|} | \mathcal{P} \in dec(P)\}$.

We will use for the sequential components of decomposed process the following *regularization rules.*

In these rules 1-3 below we will use the operations of union $\cup$ and intersection $\cap$ extended in the following way:

$$(A\|) \circ B = (A \circ B)\|$$
$$(\|A) \circ B = \|(A \circ B)$$
$$(\mathcal{A}_1, \mathcal{A}_2) \circ B = (\mathcal{A}_1 \circ B, \mathcal{A}_2 \circ B)$$

where $B \subseteq \alpha$, $\mathcal{A}_1$ and $\mathcal{A}_2$ - communication information of type (*), and $\circ \in \{\cap, \cup\}$.
For example, $(A_2\|, \|A_2) \cup B = ((A_1 \cup B)\|, \|(A_2 \cup B))$

So, the *regularization rules* are the following:

1. $$((\overset{n}{\underset{i=1}{\cup}} (\mathcal{P}_i)_{\mathcal{A}_i})_{\mathcal{A}} = \overset{n}{\underset{i=1}{\cup}} (\mathcal{P}_i)_{\mathcal{A}_i, \mathcal{A} \cap \alpha(\mathcal{P}_i)}$$

   We apply this rule when there are several occurrences of concurrency operator in the decomposed process.

   For example, let us consider the process $P = ((a\|b)\|(a;c))$. Then $dec(P) = ((a)_{\emptyset\|})_{\{a\}\|} \cup ((b)_{\|\emptyset})_{\{a\}\|} \cup (a;c)_{\|\{a\}} = (a)_{\emptyset\|,\{a\}\|} \cup (b)_{\|\emptyset,\emptyset\|} \cup (a;c)_{\|\{a\}}$.

2. $$(\mathcal{P})_{\mathcal{A}}; Q = (\mathcal{P}; Q)_{\mathcal{A} \cup \alpha(Q)}$$

   For example, let us consider the process $P = (a\|b);c$.
   Then $dec(P) = (a_{\emptyset\|};c) \cup (b_{\|\emptyset};c)$

Using the rule 2, we have

$a_{\emptyset\|}; c = (a; c)_{\{c\}\|}$

$b_{\|\emptyset}; c = (b; c)_{\|\{c\}}$

Thus, $dec(P) = (a; c)_{\{c\}\|} \cup (b; c)_{\|\{c\}}$

Note, that the set of sequential components for processes $((a\|b); c)$ and $((a; c)\|(b; c))$ are the same, i.e.

$dec((a\|b); c) = dec((a; c)\|(b; c))$,

and that it is in correspondence with intuitive understanding their behaviours.

3. $$(\mathcal{P})_{\mathcal{A}} \triangledown (\mathcal{Q})_{\mathcal{B}} = (\mathcal{P} \triangledown \mathcal{Q})_{\mathcal{A} \cup \alpha(\mathcal{Q}), \mathcal{B} \cup \alpha(\mathcal{P})}.$$

For example, let us consider the process $P = (a\|b)\triangledown c$.

Then $dec(P) = (a_{\emptyset\|}\triangledown c) \cup (b_{\|\emptyset}\triangledown c)$.

Using the rule 3, we have

$a_{\emptyset\|}\triangledown c = (a\triangledown c)_{\{c\}\|}$

$b_{\|\emptyset}\triangledown c = (b\triangledown c)_{\|\{c\}}$

Here we also used the obvious simplification rule: $(\mathcal{P})_{\mathcal{A},\emptyset} = (\mathcal{P})_{\mathcal{A}}$.

Thus, $dec(P) = (a\triangledown c)_{\{c\}\|} \cup (b\triangledown c)_{\|\{c\}}$.

Note that the set of sequential components for processes $((a\|b)\triangledown c)$ and $(a\triangledown c)\|(b\triangledown c)$ are the same, i.e.

$dec(a\|b)\triangledown c) = dec((a\triangledown c)\|(b\triangledown c)$,

and it meets intuitive understanding the behaviours of these processes.

Before defining operational Petri net semantics for $SAFP_2$, we need some more additional notions and definitions.

(i) Let $\mathcal{A}$ be a communication information of type (*). Then we denote by $\alpha(\mathcal{A})$ the set of actions occurring in $\mathcal{A}$, defined by the following induction rules:

$\alpha(\emptyset) = \emptyset$;

$\alpha(B\|) = \alpha(\|B) = B$

$\alpha(\mathcal{A}_1, \mathcal{A}_2) = \alpha(\mathcal{A}_1) \cup \alpha(\mathcal{A}_2)$

(ii) Let us denote by $\mathcal{A} \restriction_{\{a\}}$ *the projection* (restriction) of communication information $\mathcal{A}$ onto the action a, defined by the following rules:

$(\mathcal{A}_1, \mathcal{A}_2) \restriction_{\{a\}} = (\mathcal{A}_1 \restriction_{\{a\}}, \mathcal{A}_2 \restriction_{\{a\}})$

$$(\|A) \restriction_{\{a\}} = \begin{cases} \|\{a\} & , \text{ if } a \in A \\ \emptyset & , \text{ otherwise} \end{cases}$$

$$(A\|) \restriction_{\{a\}} = \begin{cases} \{a\}\| & , \text{ if } a \in A \\ \emptyset & , \text{ otherwise} \end{cases}$$

In such a way, we extract from the whole communication information $\mathcal{A}$ the subpart which describes the synchronization structure for the action a.

(iii) A set of communication informations $\{\mathcal{A}_1, \ldots \mathcal{A}_n\}$ is called *a complete system for action a*, iff

$$(\forall i : a \in \alpha(\mathcal{A}_i))\&(\exists P \in \mathrm{SAFP}_2 : dec(P) = \cup_{i=1}^{n}((a)_{\mathcal{A}_i \restriction \{a\}}).$$

In such a way, we distinguish <u>all</u> the sequential components which are required to be synchronized by execution of action a.

4.4 Operational Petri net semantics for SAFP_2

The operational Petri net semantics is a mapping which assigns to each process P of SAFP_2 the abstract Petri net:

$$O[\![P]\!] = (Seq, \longrightarrow, dec(P))/\approx_o$$

Following Plotkin's structural approach to operational semantics [Pl], the underlying concrete Petri net takes Seq as a set of local states and the initial marking is given by $dec(P)$, and transition relation $\rightarrow$ is defined by structural induction on process formulae.

Thus for set of sequential components $\mathcal{P}, \mathcal{Q}$ and action a we state the transitions:

$$\mathcal{P} \xrightarrow{a} \mathcal{Q}$$

either explicitly or inductively by rules of the form:

$$\frac{T_1, \ldots, T_m}{T'_1, \ldots, T'_n} \text{ where } \ldots$$

stating that if $T, \ldots, T_m$ are transitions satisfying the condition "..." then $T'_1, \ldots T'_n$ are also transitions.

Now, we present the transition relation $\rightarrow$ of $O[\![P]\!]$ with its inductive definition of local transitions.

$$\mathcal{P} \xrightarrow{a} \mathcal{Q}$$

1. *Sequential composition*

   (i) $\{a\} \xrightarrow{a} \{\nu\}$, where $a \neq \delta$,

   i.e. a process consisting of elementary action a after accomplishing the action a successfully terminates.

   (ii) $\dfrac{\mathcal{P} \xrightarrow{a} \mathcal{P}'}{\mathcal{P};Q \xrightarrow{a} < \mathcal{P}';Q >}$, where $< \mathcal{P}';Q >= \begin{cases} dec(Q), \text{ if } \mathcal{P}' = \nu \\ \mathcal{P}';Q, \text{ otherwise,} \end{cases}$

   i.e. if in sequential composition, the first "decomposed" process part terminates successfully then the decomposition of the second process part takes place.

2. *Concurrency*

   (i) *Synchrony*

   $\dfrac{\mathcal{P}_1 \xrightarrow{a} \mathcal{P}'_1, \ldots \mathcal{P}_n \xrightarrow{a} \mathcal{P}'_n}{\cup_{i=1}^{n}(\mathcal{P}_i)_{\mathcal{A}_i} \xrightarrow{a} \cup_{i=1}^{n}(\mathcal{P}'_i)_{\mathcal{A}_i}}$, where $\{\mathcal{A}_1, \ldots \mathcal{A}_n\}$ is a complete system for action a

   The requirements of "completeness" guarantees that the action a will be accomplished only in the case when all the sequential components containing this action are ready to perform it.

   (ii) *Asynchrony*

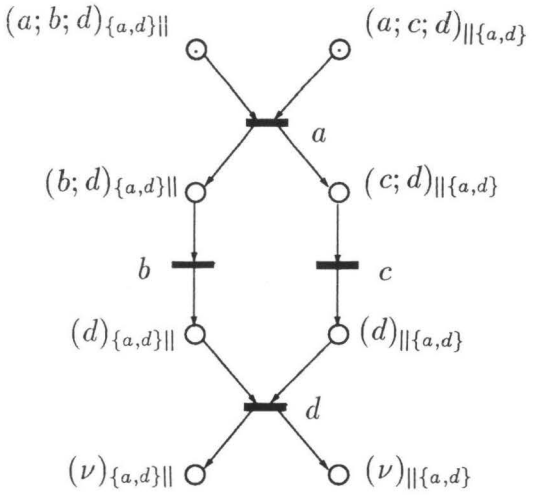
   $\dfrac{\mathcal{P} \xrightarrow{a} \mathcal{P}'}{(\mathcal{P})_{\mathcal{A}} \xrightarrow{a} (\mathcal{P}')_{\mathcal{A}}}$, where $a \notin \alpha(\mathcal{A})$

3. *Alternative*

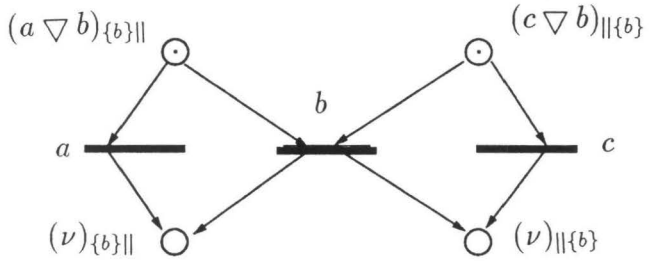
   $\dfrac{\mathcal{P} \xrightarrow{a} \mathcal{P}'}{(\mathcal{P} \triangledown \mathcal{Q}) \xrightarrow{a} \mathcal{P}'}$

4.5 Examples

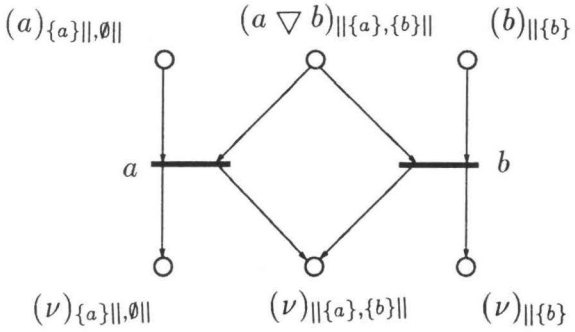
1.) $P_1 = ((a;b)\|(a;c));d)$
$dec(P_1) = (a;b;d)_{\{a,d\}\|} \cup (a;c;d)_{\|\{a,d\}}$



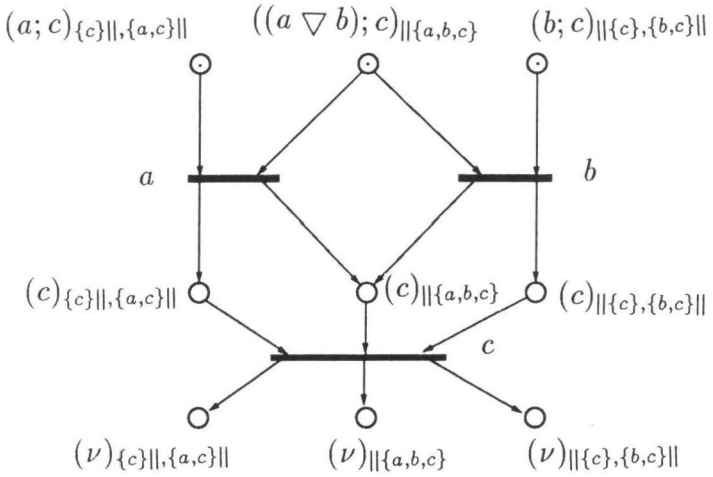
2.) $P_2 = (a\|c)\triangledown b$
$dec(P_2) = (a\triangledown b)_{\{b\}\|} \cup (c\triangledown b)_{\|\{b\}}$



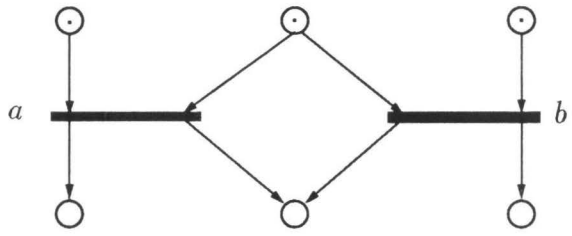
3.) $P_3 = ((a\|(a\triangledown b))\|b)$
$dec(P_3) = (a)_{\{a\}\|,\emptyset\|} \cup (a\triangledown b)_{\|\{a\},\{b\}\|} \cup (b)_{\|\{b\}}$



4.) $P_4 = (((a\|b)\|(a \triangledown b)); c)$
$dec(P_4) = (a; c)_{\{c\}\|,\{a,c\}\|} \cup (b; c)_{\|\{c\},\{b,c\}\|} \cup ((a \triangledown b); c)_{\|\{a,b,c\}}$



Please, notice that the "concrete" Petri nets for processes P_3 and P_4 above are o-equivalent and can be represented by the following "abstract" Petri net:



Thus $O[\![P_3]\!] = O[\![P_4]\!]$,

4.5 Full abstractness of denotational semantics for SAFP$_2$ w.r.t. operational net semantics

In this section, the interrelation between denotational semantics for SAFP$_2$ based on posets with non-actions and operational Petri net semantics are established, using the result about interrelation between processes of SAFP$_2$ and Petri nets specified by AFP$_o$ formulae.

Lemma 4.1
Let P and Q be processes of SAFP$_2$ and C [] be a context.

If $\forall \mathcal{C}[\,] : O[\![\mathcal{C}[P]]\!] = O[\![C[Q]]\!]$ then $\alpha(P) = \alpha(Q)$,

i.e. if operational net semantics does not distinguish the processes P and Q in any context $\mathcal{C}[\,]$, then processes P and Q have the same action basis.

Proof.
Let $a \in \alpha(P)$. there are two possibilities:

(i) either the action a occurs in the net $O[\![P]\!]$ as a transition (and it means that a can fire in the net $O[\![N]\!]$)

(ii) or the action a does not occur in the net $O[\![P]\!]$ (because of some contradiction in process formula which leads to impossibility to accomplish the action a for any possible process runs).

In the case (i), by lemma condition we have $O[\![P]\!] = O[\![Q]\!]$, and have the action a occurs in the net $O[\![Q]\!]$ as well.

Therefore, $a \in \alpha(Q)$.

In the case (ii), let us consider the processes $(P\|a)$ and $(Q\|a)$.
It is clear, that the action a can not occur (can not be performed) in the $O[\![P\|a]\!]$ by the same reason why it can't occur in the net $O[\![P]\!]$.
However, if $a \notin \alpha(Q)$, then the action a can be successfully performed in the net $O[\![Q\|a]\!]$, but by lemma condition $O[\![P\|a]\!] = O[\![Q\|a]\!]$.
Hence $a \in \alpha(Q)$. ■

Lemma 4.2
Let P be a process of SAFP_2 and let $N = \psi(P)$ be AFP_o net (see section 3.4 for definition of ψ.
Then $N \approx_o O[\![P]\!]$

Proof.
The net N built by means of AFP_o operations and the net $O[\![P]\!]$, in fact, have the same structure, excepting some redundant places and not reachable places and transitions. We will prove the lemma by induction on the net structure.

(i) For $P = a$ and $N = a$ the lemma holds trivially.

(ii) Let $P = (P_1; P_2)$ and correspondingly $N = (N_1; N_2)$, and by induction conjecture $N_1 \approx_o O[\![P_1]\!]$ and $N_2 \approx_o O[\![P_2]\!]$.
We have to show that $(N_1; N_2) \approx_o O[\![P_1; P_2]\!]$.

By definition of operational net semantics (transition rule for sequential composition) if there exist a run (maximal O-subnet) which successfully terminates then $O[\![P_1; P_2]\!] = O[\![P_1]\!]; O[\![P_2]\!]$ and since $N_1 \approx_o O[\![P_1]\!]$ and $N_2 \approx_o O[\![P_2]\!]$:

$(N_1; N_2) \approx_o O[\![P_1]\!]; O[\![P_2]\!] = O[\![P_1; P_2]\!]$.

If all runs (in fact, maximal O-subnets) of $O[\![P_1]\!]$ terminate unsuccessfully then $O[\![P_1; P_2]\!] = O[\![P_1]\!]$, but in this case all maximal O-subnets of N_1 have unsuccessfully termination (by definition of $\approx_o$ and conjecture that $O[\![P_1]\!] \approx_o N_1$), and the set of maximal O-subnets of $(N_1; N_2)$ consists of maximal O-subnets of N_1.

Thus $(N_1; N_2) \approx_o N_1 \approx_o O[\![P_1]\!] \approx_o O[\![P_1; P_2]\!]$

(iii) Let $P = (P_1 \triangledown P_2)$ and $N = (N_1 \triangledown N_2)$ correspondingly, and by induction conjecture $N_1 \approx_o O[\![P_1]\!]$, $N_2 \approx_o O[\![P_2]\!]$.

Using the definition of transition rule for alternative $\triangledown$ in operational net semantics, we have $O[\![P_1 \triangledown P_2]\!] = O[\![P_1]\!] \triangledown O[\![P_2]\!]$.

Hence, $N_1 \triangledown N_2 \approx_o O[\![P_1]\!] \triangledown O[\![P_2]\!] = O[\![P_1 \triangledown P_2]\!]$.

(iv) Let $P = (P_1 \| P_2)$ and $N = (N_1 \| N_2)$ correspondingly, and by induction conjecture $N_1 \approx_o O[\![P_1]\!]$, $N_2 \approx_o O[\![P_2]\!]$.

By definition of operational net semantics (transition rules for concurrency), if $\alpha(P_1) \cap \alpha(P_2) = \emptyset$ then

$$(N_1 \| N_2) \approx_o O[\![P_1]\!] \| O[\![P_2]\!] = O[\![P_1 \| P_2]\!].$$

If $\alpha(P_1) \cap \alpha(P_2) \neq \emptyset$ and in the united superposed net $O[\![P_1]\!] \| O[\![P_2]\!]$, all the transitions (actions) from $\alpha(P_1) \cap \alpha(P_2)$ are reachable then $(N_1 \| N_2) \approx_o O[\![P_1]\!] \| O[\![P_2]\!] = O[\![P_1 \| P_2]\!]$.

If $\alpha(P_1) \cap \alpha(P_2) \neq \emptyset$ and there exists transition $t \in \alpha(P_1) \cap \alpha(P_2)$ which never can fire in the net $O[\![P_1]\!] \| O[\![P_2]\!]$ (it means that t does not occur in any maximal O-subnet of $O[\![P_1]\!] \| O[\![P_2]\!]$ as well) then $O[\![P_1 \| P_2]\!] \approx_o O[\![P_1]\!] \| O[\![P_2]\!]$ and the net structure does not contain (in comparing with the net in right side) the places and transitions which are not reachable.

Thus, $(N_1 \| N_2) \approx_o O[\![P_1]\!] \| O[\![P_2]\!] \approx_o O[\![P_1 \| P_2]\!]$. ■

The following lemma is corollary of Lemma 4.2.

Lemma 4.3

$$\mathcal{D}_o[\![P]\!] \approx_o \mathcal{D}_o[\![Q]\!] \quad \Leftrightarrow \quad O[\![P]\!] = O[\![Q]\!]$$

Theorem 4.1
Denotational semantics $\mathcal{D}_2$ for process of SAFP_2 is fully abstract w.r.t. operational Petri net semantics, i.e.

$$\mathcal{D}_2[\![P]\!] = \mathcal{D}_2[\![Q]\!] \quad \Leftrightarrow \quad \forall \mathcal{C}[\,] : O[\![C[P]]\!] = O[\![C[Q]]\!].$$

Proof.

$\Rightarrow$

$\mathcal{D}_2[\![P]\!] = \mathcal{D}_2[\![Q]\!] \Rightarrow \forall \mathcal{C}[\,] : \mathcal{D}_2[\![C[P]]\!] = \mathcal{D}_2[\![C[Q]]\!] \Rightarrow$
by Theorem 3.2 $\forall \mathcal{C}[\,] : \mathcal{D}_o[\![C[P]]\!] \approx_o \mathcal{D}_o[\![C[Q]]\!] \Rightarrow$
by Lemma 4.3 $\forall \mathcal{C}[\,] : O[\![\mathcal{C}[P]]\!] = O[\![C[Q]]\!]$.

$\Leftarrow$

$\forall \mathcal{C}[\,] : O[\![\mathcal{C}[P]]\!] = O[\![C[Q]]\!] \Rightarrow$ by Lemma 4.1 $\alpha(P) = \alpha(Q)$
$\Leftarrow$ by Lemma 4.3 $\mathcal{D}_o[\![P]\!] \approx_o \mathcal{D}_o[\![Q]\!]$ and $\alpha(P) = \alpha(Q)$
$\Leftarrow$ by Theorem 3.2 $\mathcal{D}_2[\![P]\!] = \mathcal{D}_2[\![Q]\!]$ ■

References

LNCS = Lecture Notes in Computer Science, Springer Verlag

[BHR84] Brookes, S.D.; Hoare, C.A.R.; Roscoe, A.D.: *A Theory of Comminicating Sequential Processes.* Journal of ACM, Vol. 31, No 3, pp. 560–599, 1984

[BouCa87] Boudol, G.; Castellani, I.: *On the semantics of Concurrency: Partial Orders and Transition System.* LNCS, Vol. 249, p. 123–137, 1987.

[Ch88] Cherkasova, L.A.: *On Models and Algebras for Concurrent Processes.* LNCS, Vol. 324, p. 27–43.

[Ch89] Cherkasova, L.A.: *Posets with Non-Actions: A Model for Concurrent Nondeterministic Processes.* Arbeitspapiere der GMD, N403, 1989.

[ChK90] Cherkasova, L.; Kotov, V.: *Descriptive and Analytical Process Algebras.* LNCS, Vol. 424, 1990

[Cz85] Czaja, L.: *Making Nets Structured and Abstracts.* LNCS, Vol. 222, p. 181–202, 1985.

[**DDM86**] Degano, P.; DeNicola, R., Montanari, U.: *A New Operational Semantics for CCS Based on Condition/Event Systems.* Nota Interna B4-42, Dept. of Computer Sciences, Univ. Pisa, 1986

[**Ho85**] Hoare, C.A.R.: *Communicating Sequential Processes.* Prentice Hall, London, 1985.

[**Ke76**] Keller, R.M.: *Formal Verification of Parallel Programs.* Comm. ACM, p. 371–384, 1976.

[**Kot78**] Kotov, V.E.: *An Algebra for Parallelism Based on Petri Nets.* LNCS, Vol. 64, p. 39–55, 1978.

[**Mil80**] Milner, R.: *Calculus of Communicating Systems.* LNCS, Vol. 92, 1980

[**Mil83**] Milner, R.: *Calculi for Synchrony and Asynchrony.* J. Theoretical. Computer Science, Vol. 25, p. 267–310, North Holland, 1983.

[**NPW81**] Nielsen, M.; Plotkin, G.; Winskel, G.: *Petri Nets, Event Structures and Domains.* J. Theoretical Computer Science, Vol. 13, p. 85–108, 1981.

[**Ol88**] Olderog, E.R.: *Operational Petri Net Semantics for CCSP.* LNCS, Vol. 266, p. 196–223, 1988.

[**Pet77**] Petri, C.A.: *Non–Sequential Processes.* GMD–ISF, Rep. 77–05, 1977.

[**Pl81**] Plotkin, G.D.: *Structured Approach to Operational Semantics.* Tech. Report DAIMI FN-19, Comp. Science Dept., Aarhus Univ., 1981

[**Pr87**] Pratt, V.R.: *Modelling Concurrency with Partial Orders.* International Journal of Parallel Programming, Vol. 15, No 1, p. 33–71, 1987.

[**Th87**] Thiagarajan, P.S.: *Elementary Net Systems.* LNCS, Vol. 254, p. 26–59.