Theoretical Computer Science 43 (1986) 213-238
North-Holland

CONCURRENT AND MAXIMALLY CONCURRENT EVOLUTION OF NONSEQUENTIAL SYSTEMS

Ryszard JANICKI *
Institute of Electronic Systems, Aalborg University Centre, DK-9000 Aalborg, Denmark

Peter E. LAUER
Department of Computer Science and Systems, McMaster University, Hamilton, Ontario L8S 4K1, Canada

Maciej KOUTNY *
Computing Laboratory, The University of Newcastle upon Tyne, Newcastle upon Tyne NE1 7RU, United Kingdom

Raymond DEVILLERS
Laboratoire d'Informatique Théorique, Université Libre de Bruxelles, B-1050 Brussels, Belgium

Communicated by R. Milner
Received March 1985
Revised October 1985

Abstract. The semantics expressed intuitively as 'execute as much as possible in parallel' is formally defined and analysed. The relation between such a maximally concurrent semantics and 'normal' concurrent semantics is developed. Necessary and sufficient criteria for the equivalence of these semantics are formulated. As an abstract model of nonsequential systems the COSY path expression formalism is used.

1. Introduction

Among various semantics of executions in nonsequential systems (see discussion in [4]) we can distinguish two widely accepted approaches. The first one, standard in the Petri net approach (cf. [3, p. 528]), may intuitively be expressed as: 'execute as possible'; this encompasses the whole range of (possibly) concurrent evolutions; from the sequential ones to the maximally concurrent ones through all the intermediate cases, any possible execution is allowed. Vector firing sequence semantics for the COSY path expressions described in [19, 28] is another good example of this

* On leave from the Institute of Mathematics, Warsaw Technical University, Warsaw, Poland. Present affiliation: Department of Computer Science and Systems, McMaster University, Hamilton, Canada.

approach. Interleaving semantics of Milner [23] and Hoare [9] have also their roots in the intuition 'execute as possible'.

The second one is usually expressed as 'execute as much as possible in parallel' or 'execute as quick as possible'. In this case we require that processes should not be lazy, and at each step of the computation, the set of instructions must be a maximal non-conflict set. The first semantics is more general but in practice the second one is sometimes more natural and easier to implement (compare [2, 5, 27]).

The computational power of the second semantics was studied by Burkhard [5] in the case of Petri nets, while Enjalbert and Michel [7] have analysed this approach in terms of temporal logic.

In this paper we try to answer the question: "When is the semantics 'execute as much as possible in parallel' equivalent to the semantics 'as possible'?".

Concurrent systems may exhibit extremely complicated behaviours and informal reasoning is not reliable enough to establish their properties, so we must use an abstract formal model. As a formal model we shall use the COSY path expression formalism [15, 19, 20], which is sufficiently wide. It has been developed to a great level of sophistication and provides a respectable number of analytic criteria which are also efficiently mechanisable.

Furthermore, a computer based environment, called BCS and based on the COSY formalism, was recently implemented (cf. [8, 16, 17]). BCS permits, among other things, the analysis of a COSY system specification by concurrent simulation. In the process of using this simulator, it became obvious that systems for which the maximally concurrent behaviour determines the full behaviour are much more easy to analyse.

In this paper we shall recall the classical formal description of the semantics 'execute as possible' and we shall define a similar formal description of the semantics 'execute as much as possible in parallel'. Necessary and sufficient criteria for the equivalence of both semantics are then formulated and proved.

Although the results of the paper are formulated in terms of the COSY path expressions theory, they may be translated into other related formalisms. For instance, all results may immediately be transformed into the language of Petri nets.

In spite of the above-mentioned twenty years of intensive research, in practice, when we analyse the dynamic properties of the Petri net specification of a real concurrent system, we are frequently forced to analyse the reachability graph of the net (if it is safe) or its reachability tree (if it is unsafe, see [25]). This is usually a very long and uphill task, often impossible without the assistance of Computer-Aided-Design Tools (like those in [13, 16, 24]), and even though we have such an assistance the task is still uneasy.

The reachability graph defined by maximally concurrent behaviour is much smaller than the one defined by the full behaviour. When a safe system contains a dozen of concurrent actions, it may be smaller—according to practical experiences of the two first authors with BCS—by a factor of ten (or even more) times. This means that the systems for which the maximally concurrent behaviour determines the full

behaviour are much more easy to analyse, particularly when we cannot use any theorem (because of the generality and/or complexity of the system to be specified).

Some results of the paper have been announced at the 4th European Workshop on Application and Theory of Petri Nets, Toulouse, 1983, and the International Seminar on Concurrency, Carnegie-Mellon University, 1984.

As a general remark, we may also mention that some of our results may be connected to the general theory of partial orders; in order to be more selfcontained, to better support intuition, and to get more constructive proofs, we shall stick to our original context however.

2. Basic COSY syntax and its standard semantics

COSY (COncurrent SYstem) is a formalism intended to simplify the study of synchronic aspects of concurrent systems, by abstracting from all aspects of systems except those which have to do with synchronisation.

A basic COSY path program is a collection of paths enclosed in **program** and **endprogram** parentheses. Essentially, a path is a regular expression enclosed by **path** and **end**, as, for instance,

$P0 =$ **program**
$P1$: **path** a; b, c **end**
$P2$: **path** $(d; e)^*$; b **end**
endprogram.

In every regular expression like the above program, the semicolon denotes sequence (concatenation) and the comma denotes mutually exclusive choice. The comma binds more strongly than the semicolon, so that the expression "a; b, c" means 'first a, then either b or c'. An expression may be enclosed in conventional parentheses with Kleene star appended, as, for instance, "$(d; e)^*$" which means that the enclosed expression may be executed zero or more times. The expression appearing between **path** and **end** is implicitly 'starred', so that a path describes a cyclic sequential subsystem. The formal description of the COSY syntax may be found, for instance, in [15, 19, 20, 28]. The papers [15, 19, 20] contain various examples of using COSY to specify real systems. The semantics of path programs can be described by means of vectors of strings (an approach initiated by Shields [28]).

With every path $P =$ **path** body **end**, we associate its set of events $\mathrm{Ev}(P)$. In the case of example $P0$, the events are: $\mathrm{Ev}(P1) = \{a, b, c\}$, $\mathrm{Ev}(P2) = \{b, d, e\}$, which also indicates how events are distributed into subsystems. As it was pointed out above, 'body' may be treated as an ordinary regular expression. The only difference is replacing "$\cup$" by ",", using ";" to denote concatenation, and assuming that a mutually exclusive choice binds more strongly than concatenation (in traditional notation the opposite convention is used). Thus, for instance, "a; b, c" is equivalent to "$a(b \cup c)$" according to traditional notation.

For every regular expression E, let $|E|$ denote the regular language described by E. For every path $P =$ **path** body **end** the language |body| is called the set of *cycles* of P and denoted by $\mathrm{Cyc}(P)$, i.e., $\mathrm{Cyc}(P) = |\mathrm{body}|$.

For example, for $P0$ we obtain $\mathrm{Cyc}(P1) = \{ab, ac\}$, $\mathrm{Cyc}(P2) = \{de\}^*\{b\}$; they represent the periods of the inherently periodic sequential subsystems, namely the single paths.

From the set $\mathrm{Cyc}(P)$ we construct the set of *firing sequences* of P, denoted $\mathrm{FS}(P)$, as follows:

$$\mathrm{FS}(P) = \mathrm{Pref}(\mathrm{Cyc}(P)^*) = \mathrm{Cyc}(P)^*\mathrm{Pref}(\mathrm{Cyc}(P)),$$

where, for every alphabet A and every language $L \subseteq A^*$,

$$\mathrm{Pref}(L) = \{x \mid (\exists y \in A^*)xy \in L\}.$$

The set $\mathrm{FS}(P)$ is the set of finite sequences of event occurrences specified by the path P. For example, for $P0$, we obtain

$$\mathrm{FS}(P1) = \{ab, ac\}^*\{\varepsilon, a\}, \qquad \mathrm{FS}(P2) = (\{de\}^*\{b\})^*\{de\}^*\{\varepsilon, d\},$$

where ε denotes the empty string.

Let us now consider a path program $P =$ **program** $P_1 \ldots P_n$ **endprogram** (or simply $P = P_1 \ldots P_n$), where the P_i's are single paths. To model the nonsequential behaviour of P, partial orders of occurrences of events will be constructed which are represented by vectors of strings.

A vector $(x_1, \ldots, x_n)$ is a possible behaviour of $P = P_1 \ldots P_n$ if each x_i (for $i = 1, \ldots, n$) is a possible firing sequence of P_i and, furthermore, if the x_i's agree about the number and order of occurrences of events they share. To formally define the set of possible behaviours or histories of P, vectors of strings are introduced together with a concatenation operation on them.

Let us consider the set $\mathrm{Ev}(P_1)^* \times \cdots \times \mathrm{Ev}(P_n)^*$. If the vectors $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ belong to the above set, their concatenation is defined as

$$(x_1, \ldots, x_n)(y_1, \ldots, y_n) = (x_1y_1, \ldots, x_ny_n).$$

Let $\mathrm{Ev}(P) = \mathrm{Ev}(P_1) \cup \cdots \cup \mathrm{Ev}(P_n)$ and, for $i = 1, \ldots, n$, let $h_i : \mathrm{Ev}(P)^* \to \mathrm{Ev}(P_i)^*$ be the erasing homomorphism (endomorphism and projection, in fact) defined from:

$$(\forall a \in \mathrm{Ev}(P))\ h_i(a) = \begin{cases} a & a \in \mathrm{Ev}(P_i), \\ \varepsilon & \text{otherwise.} \end{cases}$$

Let $\_ : \mathrm{Ev}(P)^* \to \mathrm{Ev}(P_1)^* \times \cdots \times \mathrm{Ev}(P_n)^*$ be the mapping defined as follows:

$$(\forall x \in \mathrm{Ev}(P)^*)\ \underline{x} = (h_1(x), \ldots, h_n(x)).$$

Conventionally we shall write x instead of $\underline{x}$ in every place where this does not lead to ambiguity. Clearly, we may have $x \neq y$ and $\underline{x} = \underline{y}$. The set $\mathrm{Vev}(P) = \{\underline{a} \mid a \in \mathrm{Ev}(P)\}$ is called the set of *vector events* of P. Note that $\underline{\mathrm{Ev}(P)^*} = \mathrm{Vev}(P)^*$.

For example, for $P0$, the vector events are

$$\mathrm{Vev}(P0) = \{\underline{a}, \underline{b}, \underline{c}, \underline{d}, \underline{e}\} = \{(a, \varepsilon), (b, b), (c, \varepsilon), (\varepsilon, d), (\varepsilon, e)\},$$

or, indicated by distribution into subsystems and "ε" replaced by blank:

$$\mathrm{Vev}(P0)=\left\{\begin{bmatrix}a\\ \end{bmatrix},\begin{bmatrix}b\\ b\end{bmatrix},\begin{bmatrix}c\\ \end{bmatrix},\begin{bmatrix}\\ d\end{bmatrix},\begin{bmatrix}\\ e\end{bmatrix}\right\}\begin{matrix}\leftarrow \mathrm{Ev}(P1)\\ \leftarrow \mathrm{Ev}(P2)\end{matrix}.$$

Essentially, the vector events indicate distribution of events to subsystems and the sharing of events ('handshake' synchronisation) by subsystems.

For $i=1,\ldots,n$, let $|_i:\mathrm{Ev}(P_1)^*\times\cdots\times\mathrm{Ev}(P_n)^*\to\mathrm{Ev}(P_i)^*$ be a projection defined standardly as: $(x_1,\ldots,x_i,\ldots,x_n)|_i=x_i$. Note that $(\forall x\in\mathrm{Ev}(P)^*)(\forall i=1,\ldots,n)\,x|_i=h_i(x)$ and

$$(\forall(x_1,\ldots,x_n)\in\mathrm{Vev}(P)^*)(\exists x\in\mathrm{Ev}(P)^*)(x_1,\ldots,x_n)=x.$$

It turns out that vectors of sequences belonging to $\mathrm{Vev}(P)^*$ can be interpreted as *partial orders of event occurrences*, so that they may be used to model the *concurrent behaviours of P*.

Let $\mathrm{occ}(P)=\mathrm{Ev}(P)\times\{1,2,3,\ldots\}$. Elements of $\mathrm{occ}(P)$ are called *event occurrences*. Every string $x\in\mathrm{Ev}(P)^*$ generates a natural *total order* $T_x\subseteq\mathrm{occ}(P)\times\mathrm{occ}(P)$. For instance, if $x=abacba$, then $T_x=(a,1)\to(b,1)\to(a,2)\to(c,1)\to(b,2)\to(a,3)$. A formal (easy) definition of T_x is left to the reader (see [10]). The *partial order* $P_x\subseteq\mathrm{occ}(P)\times\mathrm{occ}(P)$ generated by $x\in\mathrm{Vev}(P)^*$ can be defined as follows (the idea follows from Szpilrajn-Marczewski [31]):

$$P_x=\bigcap_{y\in\mathrm{Ev}(P)^*\,\&\,y=x}T_y.$$

More details on this subject may be found in [10, 29].

For example, in $P0$, $\boldsymbol{adbead}\in\mathrm{Vev}(P0)^*$ and the partial order $P_{\boldsymbol{adbead}}$ is shown in Fig. 1.

The set of all possible behaviours or histories of P, the *vector firing sequences* of P, denoted by $\mathrm{VFS}(P)$, is defined by

$$\mathrm{VFS}(P)=(\mathrm{FS}(P_1)\times\cdots\times\mathrm{FS}(P_n))\cap\mathrm{Vev}(P)^*.$$

The set $\mathrm{FS}(P_1)\times\cdots\times\mathrm{FS}(P_n)$ in the definition of $\mathrm{VFS}(P)$ guarantees that each string component x_i of a history $x=(x_1,\ldots,x_n)\in\mathrm{VFS}(P)$ is a firing sequence of the path

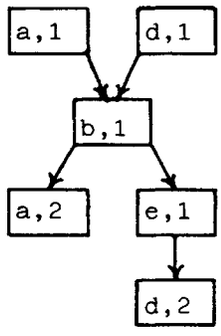


Fig. 1.

P_i, and the set $\mathrm{Vev}(P)^*$ guarantees that all these firing sequences agree on the number and order of occurrences of events they share.

The set $\mathrm{VFS}(P)$ can be treated as a formal description of the execution semantics: 'execute as possible'; this will be further consolidated by some of the following discussions (see, for instance, Theorem 3.3).

Let $r: \mathrm{Ev}(P) \to 2^{\{1,2,\ldots,n\}}$ be the function given by

$$(\forall a \in \mathrm{Ev}(P))\ r(a) = \{i \mid a \in \mathrm{Ev}(P_i)\}$$

(note that r is a well-defined 'resource association function', according to [11]) and let $\mathrm{ind} \subseteq \mathrm{Ev}(P) \times \mathrm{Ev}(P)$ be the following relation:

$$(\forall a, b \in \mathrm{Ev}(P))\ (a, b) \in \mathrm{ind} :\Leftrightarrow r(a) \cap r(b) = \emptyset.$$

The relation ind is called the *independency relation.* Note that $(a, b) \in \mathrm{ind} \Leftrightarrow (a \neq b \ \&\ \boldsymbol{ab} = \boldsymbol{ba}) \Leftrightarrow ((\forall i)\boldsymbol{a}|_i \neq \varepsilon \Rightarrow \boldsymbol{b}|_i = \varepsilon) \Leftrightarrow ((\forall i) a \notin \mathrm{Ev}(P_i)$ or $b \notin \mathrm{Ev}(P_i))$.

Since the paths correspond to the sequential subsystems of the whole system, two independent events have no common sequential constraint, so that independency may be viewed as a potential concurrency relation: only independent events may occur concurrently. However, independent events may not always occur concurrently; it may even happen that they may never occur concurrently at all: they have to be simultaneously 'enabled' at some point, this will be developed in Section 3.

The formal model of behaviour allows us to speak formally of dynamic properties of systems specified by a path program $P = P_1 \ldots P_n$.

We say that $P = P_1 \ldots P_n$ is *deadlock-free* if and only if

$$(\forall \boldsymbol{x} \in \mathrm{VFS}(P))(\exists a \in \mathrm{Ev}(P))\ \boldsymbol{xa} \in \mathrm{VFS}(P),$$

that is, every history $\boldsymbol{x}$ may be continued.

We say that $P = P_1 \ldots P_n$ is *adequate* if and only if

$$(\forall \boldsymbol{x} \in \mathrm{VFS}(P))(\forall a \in \mathrm{Ev}(P))(\exists \boldsymbol{y} \in \mathrm{Vev}(P)^*)\boldsymbol{xya} \in \mathrm{VFS}(P),$$

that is, every history $\boldsymbol{x}$ of P may be continued, eventually enabling every event in P. Adequacy is a property akin to the absence of partial system deadlock. More details can be found in [15, 19, 28].

The semantics of COSY path programs may also be expressed in terms of labelled Petri nets (cf. [3, p. 295]). The current net semantics of path programs is obtained by translating each sequential path component into a labelled state machine represented as a net, i.e., representing transitions by boxes. For example, the paths

P_1: **path** a; b; a **end**, $\qquad$ P_2: **path** a, c; d **end**

would individually give rise to the nets shown in Fig. 2. Once the nets corresponding to the individual paths have been obtained, for example, two nets called N_1 and N_2, one applies a composition rule denoted by "$\oplus$" to the two nets, written $N_1 \oplus N_2$, constructed from N_1 and N_2 by the identification of transitions with the same label.

We may now give the construction of $N_1 \oplus N_2$ from nets N_1 and N_2 and illustrate it with the two example paths above:

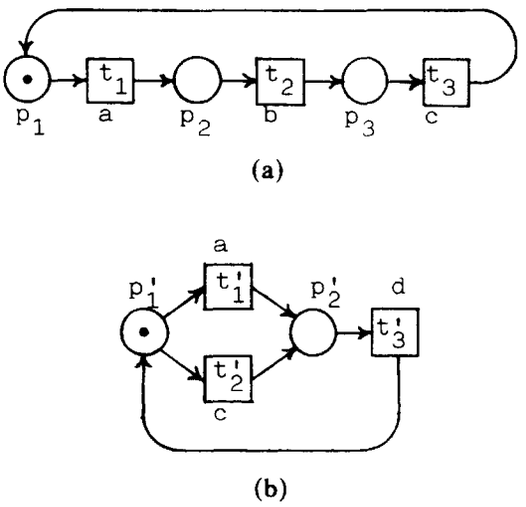


Fig. 2. (a). Net of P_1 (two transitions have the same label). (b) Net of P_2.

(1) The set of places of $N_1 \oplus N_2$ is the set-theoretic union of the sets of places of N_1 and N_2 (they are supposed disjoint!) with inherited markings.

(2) Suppose t is a transition in either N_1 or N_2 such that no transition in the other net is labelled with the label of t, then $N_1 \oplus N_2$ contains a transition (t), with the same label as t, whose input and output places are the same as those of t (recall (1)).

(3) Suppose t_1 and t_2 are transitions of N_1 and N_2 respectively, with the same label, then $N_1 \oplus N_2$ contains a transition (t_1, t_2) with the same label as t_1 and t_2 and whose set of input (respectively output) places is the union of the sets of input (respectively output) places of t_1 and t_2.

The operation $\oplus$ may be shown to be *commutative* and *associative* (up to isomorphism).

If $P = P_1 \ldots P_n$ and N_i is the marked labelled state machine net associated with P_i, then the net associated with P is defined to be $N_1 \oplus \cdots \oplus N_n$.

The result of applying these rules to $P = P_1 P_2$, where P_1 and P_2 are our example paths, then is the net shown in Fig. 3.

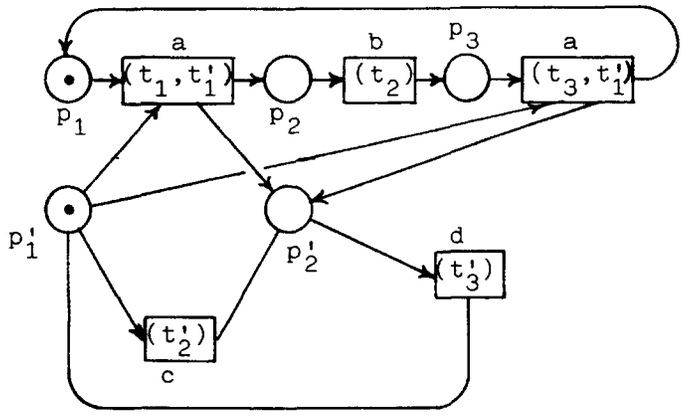


Fig. 3. N = (net of P_1) $\oplus$ (net of P_2) corresponding to the path P = **path** a; b; a **end path** a, c; d **end**.

More details about the formal correspondence between path programs and Petri nets may be found in [1, 18].

One may also prove that the algebra of vector events is equivalent to the algebra of Mazurkiewicz traces (a model of concurrent behaviours generated by Petri nets, see [14, 21, 22]) in the sense that the partial orders defined by a vector x and a trace $[x]_{\text{ind}}$ are exactly the same and the vector events monoid and trace monoid are isomorphic (see, for instance, [29]).

3. The problem of maximally concurrent evolution

Let us start with the following example (see [15]):

$P =$ **path** $a, c; d$ **end**
path $b; c, d$ **end**.

At the beginning we are able to perform actions a and b; as they are independent, we may not only initially execute a or b, but also both of them concurrently in one 'step'. In the latter case, after the concurrent performance of a and b, only action d is enabled; after its performance we are again in the initial situation. Note that in this way, if we always choose to do a and b concurrently, we are never able to perform the action c. On the other hand, we may start with the performance of the action b only and then we are able to perform c next. This example shows that doing two independent and simultaneously enabled actions in one step is not the same as doing them in either order but sequentially, that is, in two steps.

This example shows that the semantics 'execute as much as possible in parallel' may not be equivalent to the semantics 'execute as possible'. Let us analyse this problem formally. First of all, we must formally define the semantics 'execute as much as possible in parallel'.

Let $P = P_1 \dots P_n$ be a path program, and let $\text{Ind}(P) \subseteq 2^{\text{Ev}(P)} - \{\emptyset\}$ be the following family of sets of events:

$$A \in \text{Ind}(P) \;:\Leftrightarrow\; ((\forall a, b \in A)\; a = b \text{ or } (a, b) \in \text{ind}) \;\&\; A \neq \emptyset.$$

In other words, elements of $\text{Ind}(P)$ are sets of pairwise independent events. If $A = \{a_1, \dots, a_k\} \in \text{Ind}(P)$, then $\boldsymbol{a}_1 \dots \boldsymbol{a}_k = \boldsymbol{a}_{i_1} \dots \boldsymbol{a}_{i_k}$ for any permutation $i_1, \dots, i_k$ so that we may write $\boldsymbol{A} = \boldsymbol{a}_1 \dots \boldsymbol{a}_k$ and $\boldsymbol{xA} = \boldsymbol{xa}_1 \dots \boldsymbol{a}_k$.[1]

For every $x \in \text{VFS}(P)$, an event $a \in \text{Ev}(P)$ is said to be *enabled* at x if and only if

$$(\forall i = 1, \dots, n)\; a \in \text{Ev}(P_i) \Rightarrow x|_i a \in \text{FS}(P_i).$$

For every $x \in \text{VFS}(P)$, a set of independent events $A \in \text{Ind}(P)$ is said to be *concurrently enabled* at x if and only if every $a \in A$ is enabled at x.

[1] Note that *not always* the convention $\boldsymbol{A}$ instead of $\underline{\boldsymbol{A}}$ can be used. If, for instance, $A = \{a, b, c\} \in \text{Ind}(P)$, $B = \{c, d\} \in \text{Ind}(P)$, $(e, c), (e, d) \in \text{ind}$, we may write $\boldsymbol{AdB}$, but we *must* write $\underline{\{a, b, c\}}\boldsymbol{d}\underline{\{c, d\}}$ and $\underline{\boldsymbol{AB} \cup \{e\}}$! In the sequel we shall use boldfaced types where possible and underlining where necessary.

Corollary 3.1. *For every* $x \in \mathrm{VFS}(P)$,

(1) $a \in \mathrm{Ev}(P)$ *is enabled at* $x \Leftrightarrow xa \in \mathrm{VFS}(P)$,

(2) $A \in \mathrm{Ind}(P)$ *is enabled at* $x \Leftrightarrow xA \in \mathrm{VFS}(P)$.

For every $x \in \mathrm{VFS}(P)$, let enabled(x) denote the family of all concurrently enabled sets of events at x. Assume also that for $x \notin \mathrm{VFS}(P)$, enabled$(x) = \emptyset$.

Any concurrently enabled set at x, $A \in \mathrm{enabled}(x)$, is said to be *maximally concurrent* if and only if it may not be extended, i.e., iff $(\forall B \in \mathrm{enabled}(x))\ A \subseteq B \Rightarrow A = B$.

For every $x \in \mathrm{VFS}(P)$, let maxenabled(x) denote the family of all maximally concurrent sets enabled at x. Of course, maxenabled$(x) \subseteq$ enabled(x).

Corollary 3.2. *For every path program* P *the following properties are equivalent*:

(1) P *is deadlock-free*,

(2) $(\forall x \in \mathrm{VFS}(P))$ enabled$(x) \neq \emptyset$,

(3) $(\forall x \in \mathrm{VFS}(P))$ maxenabled$(x) \neq \emptyset$.

Let $S, C, M \subseteq \mathrm{Vev}(P)^* \times \mathrm{Vev}(P)^*$ be the following relations:

$$xSy \ :\Leftrightarrow\ (\exists a \in \mathrm{Ev}(P))\ \{a\} \in \mathrm{enabled}(x)\ \&\ y = xa,$$

$$xCy \ :\Leftrightarrow\ (\exists A \in \mathrm{enabled}(x))\ y = xA,$$

$$xMy \ :\Leftrightarrow\ (\exists A \in \mathrm{maxenabled}(x))\ y = xA.$$

The relations S, C, M are called respectively: *the sequential reachability in one step, the concurrent reachability in one step,* and *the maximally concurrent reachability in one step.*

Theorem 3.3

$$\mathrm{VFS}(P) = \{x \mid \varepsilon S^* x\} = \{x \mid \varepsilon C^* x\}.$$

Proof. By the definition, we have $S^* = C^*$; then $\{x \mid \varepsilon S^* x\} = \{x \mid \varepsilon C^* x\}$. Furthermore, $\mathrm{VFS}(P) = (\mathrm{FS}(P_1) \times \cdots \times \mathrm{FS}(P_n)) \cap \mathrm{Vev}(P)^*$ and from the definition of S we obtain

(1) $\varepsilon \in \{x \mid \varepsilon S^* x\} \cap (\mathrm{FS}(P_1) \times \cdots \times \mathrm{FS}(P_n)) \cap \mathrm{Vev}(P)^*$,

(2) if $y \in \{x \mid \varepsilon S^* x\} \cap (\mathrm{FS}(P_1) \times \cdots \times \mathrm{FS}(P_n)) \cap \mathrm{Vev}(P)^*$, then

$$ya \in \{x \mid \varepsilon S^* x\} \ \Leftrightarrow\ ya \in (\mathrm{FS}(P_1) \times \cdots \times \mathrm{FS}(P_n)) \cap \mathrm{Vev}(P)^*.$$

Thus $\{x \mid \varepsilon S^* x\} = (\mathrm{FS}(P_1) \times \cdots \times \mathrm{FS}(P_n)) \cap \mathrm{Vev}(P)^*$, which ends the proof. □

The above result states that VFS is fully characterisable by the relation C (as well as by the relation S) so that VFS(P) with C indeed models the concurrent evolution of the system 'execute as possible'.

It may also be noticed that $S^* = C^*$ is exactly the partial order on VFS(P) defined by the prefix relation: if $x, y \in \mathrm{VFS}(P)$,

$$xC^*y \Leftrightarrow ((\exists z \in \mathrm{Vev}(P)^*)\, y = xz) \Leftrightarrow ((\forall i)(\exists z \in \mathrm{Ev}(P_i)^*)\, y|_i = x|_i z)$$

(compare [29]).

The relation M is that mathematical object which represents the maximally concurrent evolution, i.e., one step under the rules of the semantics 'execute as much as possible in parallel'. Let us define $\mathrm{VMFS}(P) = \{x \mid \varepsilon M^* x\}$. The set VMFS($P$) represents all histories that may be reached by a maximally concurrent evolution of the system (the *vector maximal firing sequences*), so it may be treated as a *formal description* of the execution semantics: 'execute as much as possible in parallel'.

For every $V \subseteq \mathrm{Vev}(P)^*$, let $\mathrm{Pref}(V) = \{x \mid (\exists y \in \mathrm{Vev}(P)^*)\, xy \in V\}$.

Corollary 3.4

$$\mathrm{Pref}(\mathrm{VMFS}(P)) \subseteq \mathrm{VFS}(P).$$

We will say that P *is completely characterised by maximally concurrent evolution* if and only if

$$\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P)).$$

This equality is a formal expression of the fact that the semantics 'execute as much as possible in parallel' and the semantics 'execute as possible' are equivalent, as will be shown later.

4. Some properties of vector maximal firing sequences

We are now going to elucidate the structure of Pref(VMFS(P)) and to prove formally that the above concept of complete characterisation by maximally concurrent evolution is justified and well-defined. We prove that VMFS(P) is a tree and that if $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$, the notions of deadlock-freeness and adequacy may be described in terms of VMFS only.

Lemma 4.1

$$(\forall x \in \mathrm{VFS}(P))(\forall y, z \in \mathrm{VMFS}(P))\ (yC^*x \,\&\, zC^*x) \Rightarrow (yM^*z \text{ or } zM^*y).$$

Proof. Define $\mathrm{prefM}(y, z) = \{v \mid v \in \mathrm{VMFS}(P) \,\&\, vM^*y\} \cap \{w \mid w \in \mathrm{VMFS}(P) \,\&\, wM^*z\}$; of course, $\varepsilon \in \mathrm{prefM}(y, z)$. If the property is not true, $y \notin \mathrm{prefM}(y, z)$ and $z \notin \mathrm{prefM}(y, z)$. Let t be a minimal element of $\mathrm{prefM}(y, z)$. Then $(\exists y' \neq z'.\mathrm{VMFS}(P) - \mathrm{prefM}(y, z))$ such that one has the pattern

$$tMy'M^*yC^*x \quad \text{and} \quad tMz'M^*zC^*x$$

(one may have $y' = y$ and/or $z = z'$), where $y' = tA$, $z' = tB$, and $A \neq B$. Assume that $a \in A - B$. As $x = ta \ldots tB \ldots$, a must occur after B in the second expression, i.e.,

$x = ta = \cdots = tBc_1 \ldots c_k a \ldots$ and a must commute with each c_i and each b in B. But then B is no longer maximally concurrent at y since one may add $\{a\}$ to it, hence, the lemma. □

As an almost immediate consequence of Lemma 4.1 we obtain the following corollary.

Corollary 4.2. (1) VFMS(P) *with the M relation is a tree* (VFS(P) *with the C relation is not*!).

(2) *Each* $x \in \text{VFS}(P)$ *has a unique greatest prefix in* VMFS(P).

(3) *For every* $x \in \text{Pref}(\text{VMFS}(P))$, *if* y *is its greatest prefix in* VMFS(P) *and if* y' *is any of its least extensions in* VMFS(P) (*i.e., if* x *is a prefix of* y', $y' \in \text{VMFS}(P)$, *and for all* $y'' \in \text{VMFS}(P)$: $(xC^*y'' \,\&\, y''C^*y') \Rightarrow y'' = y')$, *then* yM^*y'.

About Corollary 4.2(3), we may mention that, to the contrary of Corollary 4.2(2), if x in VMFS(P), it may happen that there is no unique extension of x in VMFS(P). For:

$P =$ **path** b, c **end**
path a **end**

and for $x = a$, there are two least VMFS extensions: ab and ac. More details on these kind of properties may be found in [6].

A path program P is said to be *M-deadlock-free* if and only if:

$$(\forall x \in \text{VMFS}(P))(\exists y \in \text{Vev}(P)^*)\ y \neq \varepsilon \ \&\ xy \in \text{VMFS}(P).$$

Corollary 4.3

$$P \text{ is M-deadlock-free} \Leftrightarrow (\forall x \in \text{VMFS}(P))\ \text{maxenabled}(x) \neq \emptyset.$$

Theorem 4.4. *If* $\text{VFS}(P) = \text{Pref}(\text{VMFS}(P))$, *then*

$$P \text{ is deadlock-free} \Leftrightarrow P \text{ is M-deadlock-free.}$$

Proof. ($\Rightarrow$): A consequence of Corollaries 3.2 and 4.3.

($\Leftarrow$): Let $x \in \text{VFS}(P)$. Since $\text{VFS}(P) = \text{Pref}(\text{VMFS}(P))$, we have $(\exists z)\ xz \in \text{VMFS}(P)$. Of course $xz \in \text{VFS}(P)$, so if $z \neq \varepsilon$, then P is not deadlocked at x. If $z = \varepsilon$, then $x \in \text{VMFS}(P)$ and there is $y \neq \varepsilon$ such that $xy \in \text{VMFS}(P) \subseteq \text{VFS}(P)$; thus, P is also not deadlocked at x. □

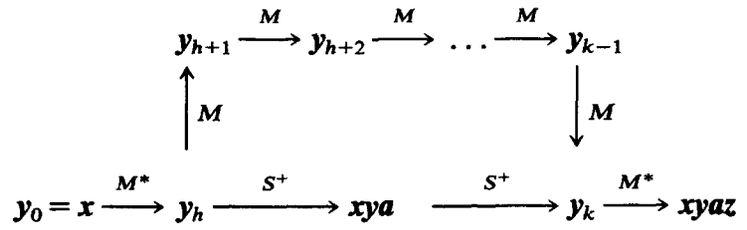
A path program P is said to be *M-adequate* if and only if

$$(\forall x \in \text{VMFS}(P))(\forall a \in \text{Ev}(P))(\exists y \in \text{Vev}(P)^*)\ xya \in \text{VMFS}(P).$$

Theorem 4.5. *If* $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$, *then*

$$P \text{ is adequate} \Leftrightarrow P \text{ is } M\text{-adequate.}$$

Proof. ($\Rightarrow$): Let $x \in \mathrm{VMFS}(P)$ and $a \in \mathrm{Ev}(p)$. Since P is adequate, $(\exists y \in \mathrm{Vev}(P)^*)\ xya \in \mathrm{VFS}(P)$, and since $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$, $(\exists z \in \mathrm{Vev}(P)^*)\ xyaz \in \mathrm{VMFS}(P)$. If $xya \in \mathrm{VMFS}(P)$, we have our result; if it is not true, we have a pattern such as the following:



where for $i = 1, \ldots, k$, $y_i = y_{i-1}A_i \in \mathrm{VMFS}(P)$. Because a belongs to one of the A_i, we may write $y_i = y_{i-1} \ldots a$, where y_{i-1} is an extension of x in $\mathrm{VMFS}(P)$.

($\Leftarrow$): Let $x \in \mathrm{VFS}(P)$, $a \in \mathrm{Ev}(P)$. Since $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$, we have $(\exists z \in \mathrm{Vev}(P)^*)xz \in \mathrm{VMFS}(P)$. Since P is M-adequate, it holds that $(\exists y)xzya \in \mathrm{VMFS}(P) \subseteq \mathrm{VFS}(P)$, and P is adequate. □

The above two theorems, showing that deadlock-freeness and adequacy may be defined from VMFS only, can be regarded as a justification of our definition of complete characterisation by maximally concurrent evolution.

5. Interlace decomposition

In this section we shall further develop the structure of $\mathrm{Vev}(P)^*$. It turns out that every $x \in \mathrm{Vev}(P)^*$ can be uniquely decomposed in a special manner and that this decomposition is a convenient tool to define necessary and sufficient conditions for the equality $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$, as will be shown in Section 6.

Let $P = P_1 \ldots P_n$ be a path program. For every $a \in \mathrm{Ev}(P)$ and $A \subseteq \mathrm{Ev}(P)$, we shall write $(a, A) \in \mathrm{ind}$ iff $(\forall b \in A)\ (a, b) \in \mathrm{ind}$. Note that, if $A \in \mathrm{Ind}(P)$:

$$(a, A) \in \mathrm{ind} \Leftrightarrow a \notin A \,\&\, A \cup \{a\} \in \mathrm{Ind}(P).$$

For every $x \in \mathrm{Vev}(P)^* - \{\varepsilon\}$, a sequence of sets $(A_1, \ldots, A_k)$, $k \geq 1$, is said to be an *interlace decomposition* (abbreviated *il-decomposition*) of x iff the following conditions are satisfied:

(1) $(\forall i = 1, \ldots, k)\ A_i \in \mathrm{Ind}(P)$,

(2) $x = A_1 \ldots A_k$,

(3) if $k > 1$, then $(\forall a \in A_i)\ (a, A_{i-1}) \notin \mathrm{ind}$ for $i = 2, \ldots, k$.

The last condition expresses the fact that the decomposition is 'left maximal', in the sense that no event may be pushed to the left, from one A_i to another, while

preserving the first two conditions, or else that, for any i, A_i is a maximal independent prefix of $A_iA_{i+1}\dots A_k$. It will be shown later (see Theorem 6.1) that problems arise precisely when, in an il-decomposition, some A_i but the last one is not in maxenabled$(A_1\dots A_{i-1})$ while the other ones are.

If $(A_1,\dots,A_k)$ is an il-decomposition of x, we shall write

$$x=^D A_1\dots A_k.$$

For instance, with the path program P described at the beginning of Section 3:

$$abac=^D \underline{\{a,b\}\{a\}\{c\}},$$

but not

$$abac=^D \underline{\{a\}\{b,a\}\{c\}}.$$

We want to prove that every $x\in \mathrm{Vev}(P)^*-\{\varepsilon\}$ has exactly one il-decomposition. Let us first notice that, from the definition, if $X=^D A_1\dots A_k$, $y=^D B_1\dots B_m$ and $(\forall b\in B_1)\ (b,A_k)\notin \mathrm{ind}$, then $xy=^D A_1\dots A_kB_1\dots B_m$. Conversely, if $x=^D A_1\dots A_k$, then $(\forall i)\ A_1\dots A_i=^D A_1\dots A_i$ and $A_i\dots A_k=^D A_i\dots A_k$. Now, we may show that in an il-decomposition the first set is uniquely determined by the considered vector.

Lemma 5.1. *If* $x=^D A_1\dots A_k$, *then*

$$\begin{aligned}A_1&=\{a\in \mathrm{Ev}(P)\mid(\forall i\in r(a))(\exists y\in \mathrm{Ev}(P_i)^*)\ x|_i=ay\}\\&=\{a\in \mathrm{Ev}(P)\mid(\exists y\in \mathrm{Vev}(P)^*)\ x=ay\}.\end{aligned}$$

Proof. Let $A=\{a\in \mathrm{Ev}(P)\mid(\forall i\in r(a))(\exists y\in \mathrm{Ev}(P_i)^*)\ x|_i=ay\}$. Suppose that $a\in A_1$. Since $x=A_1x'$ and $A_1\in \mathrm{Ind}(P)$, we have $x=az$. Hence, $a\in A$ and $A_1\subseteq A$. If $A_1\neq A$, let $a\in A-A_1$ and let j be the index of the first A_i containing a, i.e., $a\in A_j$ and $a\notin A_i$ for $i=1,\dots,j-1$ $(2\leqslant j\leqslant k)$. This means that there exists $b\in A_{j-1}$, $b\neq a$ and $(a,b)\notin \mathrm{ind}$. Therefore, if $i\in r(a)\cap r(b)$, then $x|_i=ybaw$. where $w\in \mathrm{Ev}(P_i)^*$ and $y\in(\mathrm{Ev}(P_i)-\{a\})^*$, a contradiction. Thus, $A-A_1=\emptyset$, which is now equivalent to $A=A_1$. The second equality is a simple consequence of the first one. □

Our existence and unicity result may now be proved by induction on the 'length' of the considered vector. Let us define the length function $l:\mathrm{Vev}(P)^*\to\{0,1,2,\dots\}$ by $l(\varepsilon)=0$, and if $y=a_1\dots a_m$, where $(\forall i)\ a_i\in \mathrm{Ev}(P)$, then $l(y)=m$.

Theorem 5.2. *Each* $x\in \mathrm{Vev}(P)^*-\{\varepsilon\}$ *has exactly one il-decomposition.*

Proof. The theorem is proved by induction on $l(x)$. Let $x\in \mathrm{Vev}(P)^*-\{\varepsilon\}$. If $l(x)=1$, then, of course, it has exactly one il-decomposition $\{a\}$. Assume that $l(x)\geqslant 2$ and that each $y\in \mathrm{Vev}(P)^*$ such that $1\leqslant l(y)<l(x)$ has exactly one il-decomposition. We

first want to prove that x has at least one il-decomposition. Let $A = \{a \in \mathrm{Ev}(P) \mid (\exists y \in \mathrm{Vev}(P)^*)\ x = ay\}$. Clearly, $A \in \mathrm{Ind}(P)$ and $x = Ay$. Since $l(y) < l(x)$, by the induction assumption, we have $y = \varepsilon$ or $y =^D B_1 \ldots B_k$.

Case 1: $y = \varepsilon$. Then $x =^D A$.

Case 2: $y \neq \varepsilon$. Let $b \in B_1$. Suppose that $(\forall a \in A)\ (a, b) \in \mathrm{ind}$. Then $x = bz$ for some $z \in \mathrm{Vev}(P)^*$ and $b \in A$, a contradiction. Consequently, we have $X =^D AB_1 \ldots B_k$. We still have to prove that the decomposition is unique. Suppose that $x =^D A_1 \ldots A_p$ and $x =^D B_1 \ldots B_m$. By Lemma 5.1, we have $A_1 = B_1$. If $p = 1$, this ends the proof. Otherwise, let $y = A_2 \ldots A_p$; thus, $y =^D A_2 \ldots A_p$, $y =^D B_2 \ldots B_m$ and, since $l(y) < l(x)$, by the induction assumption, we obtain $A_i = B_i$ for $i = 2, \ldots, p$ and $p = m$. □

We are now going to present some additional properties of il-decomposition. Those properties will show the connections with some other notions of our approach.

First, the prefix relation corresponds to an inclusion property.

Theorem 5.3. *If* $x, y, z \in \mathrm{Vev}(P)^*$, $y = xz, x =^D A_1 \ldots A_k$ *and* $y = B_1 \ldots B_m$, *then* $k \leq m$ *and* $A_i \subseteq B_i$ *for* $i = 1, \ldots, k$. *Moreover, if* $x, y \in \mathrm{VFS}(P)$ *and* $A_k \in \mathrm{maxenabled}(A_1 \ldots A_{k-1})$, *then* $A_i = B_i$ *for* $i - 1, \ldots, k$ (*with* $A_1 \ldots A_{k-1} = \varepsilon$ *if* $k = 1$).

Proof. Let us first consider the simple case where $y = xa$ for some $a \in \mathrm{Ev}(P)$. We have to consider three cases.

Case 1: $(a, A_i) \in \mathrm{ind}$ for $i = 1, \ldots, k$. Then $y =^D B_1 \ldots B_k$, where $B_1 = A_1 \cup \{a\}$ and $B_i = A_i$ for $i \neq 1$.

Case 2: $(a, A_k) \in \mathrm{ind}$ and $(a, A_j) \notin \mathrm{ind}$ for some $j < k$. Let j be the last such index, i.e., $(a, A_j) \notin \mathrm{ind}$ and $(a, A_i) \in \mathrm{ind}$ for $i = j+1, \ldots, k$. Then $y =^D B_1 \ldots B_k$, where $B_{j+1} = A_{j+1} \cup \{a\}$ and $B_i = A_i$ for $i \neq j+1$.

Case 3: $(a, A_k) \notin \mathrm{ind}$. Then $y = A_1 \ldots A_k \underline{\{a\}}$.

The general case $y = xz = xz_1 \ldots z_r$, where $(\forall i)\ z_i \in \mathrm{Ev}(P)$, results from r successive applications of the simple case. Moreover, if $x, y \in \mathrm{VFS}(P)$ and $A_k \in \mathrm{maxenabled}(A_1 \ldots A_{k-1})$, let $z =^D C_1 \ldots C_l$; $(\forall c \in C_1)\ (c, A_k) \notin \mathrm{ind}$, so that $y = xz =^D A_1 \ldots A_k C_1 \ldots C_l$. □

The il-decomposition is connected to the maxenabled property.

Theorem 5.4. *Let* $x = A_1 \ldots A_k \in \mathrm{VFS}(P) - \{\varepsilon\}$, *and*

(1) $A_k \in \mathrm{Ind}(P)$,

(2) $k \geq 2 \Rightarrow (\forall i = 1, \ldots, k-1)\ A_i \in \mathrm{maxenabled}(A_1 \ldots A_{i-1})$, *where* $A_1 \ldots A_{i-1} = \varepsilon$ *if* $i = 1$.

Then $x =^D A_1 \ldots A_k$.

Proof. Clearly, $(\forall y)\ \mathrm{maxenabled}(y) \subseteq \mathrm{Ind}(P)$. Suppose that for some $j \in \{2, \ldots, k\}$ and $a \in A_j$ we have $(a, A_{j-1}) \in \mathrm{ind}$. Thus, $A_{j-1} \cup \{a\} \in \mathrm{enabled}(A_1 \ldots A_{j-2})$, a contradiction. □

The property is even stronger in VMFS(P).

Theorem 5.5. *Let* $x = {}^D A_1 \dots A_k \in \mathrm{VMFS}(P) - \{\varepsilon\}$. *Then* $A_i \in \mathrm{maxenabled}(A_1 \dots A_{i-1})$ *for* $i = 2, \dots, k$, *and* $A_1 \in \mathrm{maxenabled}(\varepsilon)$.

Proof. Since $x \in \mathrm{VMFS}(P)$, let $\varepsilon = x_0 M B_1 = x_1 M x_1 B_2 = x_2 M \dots M x_{m-1} B_m = x_m = x$. By definition, $B_i \in \mathrm{maxenabled}(B_1 \dots B_{i-1})$ for $i = 1, \dots, m$. From Theorem 5.4 it follows that $x = {}^D B_1 \dots B_m$, and by Theorem 5.2, $A_i = B_i$ for $i = 1, \dots, k$ and $k = m$. □

Finally, we may connect the il-decomposition to the VMFS-prefixes.

Theorem 5.6. *If* $x \in \mathrm{VMFS}(P)$, xC^*y *and* $y = {}^D A_1 \dots A_k$, *then* $x = {}^D A_1 \dots A_j$ *for some* $j \leq k$.

Proof. Let $y = xz$, $x = {}^D A_1 \dots A_j$ and $z = {}^D B_1 \dots B_k$. By Theorem 5.5, $A_j \in \mathrm{maxenabled}(A_1 \dots A_{j-1})$ so that $(\forall b \in B_1)\ (b, A_j) \in \mathrm{ind}$ and $y = {}^D A_1 \dots A_j B_1 \dots B_k$. □

6. Necessary and sufficient conditions for complete characterisation by maximally concurrent evolution

On the basis of the results from our previous sections we can fully characterise the class of path programs possessing the property $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$.

Let $P = P_1 \dots P_n$ be an arbitrary path program. Let us define a function $m : \mathrm{Vev}(P)^* \to \{0, 1, 2, \dots\}$, by $m(\varepsilon) = 0$, and $(\forall x \in \mathrm{Vev}(P)^* - \{\varepsilon\})\ m(x) = k$ iff $x = {}^D A_1 \dots A_k$. From Theorem 5.2 it follows that $m(x)$ is well-defined.

Theorem 6.1. *The following properties are equivalent*:

(1) $\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$.

(2) $(\exists x \in \mathrm{VFS}(P))\ x = {}^D A_1 \dots A_k$ & $k \geq 2$ & $A_{k-1} \notin \mathrm{maxenabled}(A_1 \dots A_{k-2})$ & $A_k \in \mathrm{maxenabled}(A_1 \dots A_{k-1})$ (*where* $A_1 \dots A_{k-2} = \varepsilon$ *if* $k = 2$).

Proof. (2⇒1): Suppose $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$ and x is as in (2). Then xC^*y for some $y \in \mathrm{VMFS}(P)$. Let $y = {}^D B_1 \dots B_m$. From Theorem 5.3, we have $m \geq k$ and $A_i = B_i$ for $i = 1, \dots, k$. On the other hand, by Theorem 5.5, we have $B_{k-1} \in \mathrm{maxenabled}(B_1 \dots B_{k-2})$, a contradiction.

(1⇒2): Let k be a number such that

$$Z = \{x \mid x \in \mathrm{VFS}(P) - \mathrm{Pref}(\mathrm{VMFS}(P)) \ \&\ m(x) = k\} \neq \emptyset.$$

Clearly, $(\forall y \in Z)\ l(y) \leq \mathrm{card}(\mathrm{Ev}(P))^k$. Let x_0 be a member of Z with maximum length, i.e., $x_0 \in Z$ and $(\forall y \in Z)\ l(x_0) \geq l(y)$. Let $x_0 = {}^D A_1 \dots A_k$ and suppose that $A_k \notin \mathrm{maxenabled}(A_1 \dots A_{k-1})$. Thus, for some $a \in \mathrm{Ev}(P)$, we have $(a, A_k) \in \mathrm{ind}$ and

$y = A_1 \dots A_k a \in \mathrm{VFS}(P)$. Since $y \in Z$ (see the proof of Theorem 5.3, Cases 1 and 2) and $l(x_0) < l(y)$, we get a contradiction so that, necessarily, $A_k \in \mathrm{maxenabled}(A_1 \dots A_{k-1})$. Because $x_0 \notin \mathrm{VMFS}(P)$, we have $A_i \notin \mathrm{maxenabled}(A_1 \dots A_{i-1})$ for some $1 \leqslant i < k$. Thus, $k \geqslant 2$ and there exists a $j < k$ such that $A_j \notin \mathrm{maxenabled}(A_1 \dots A_{j-1})$ and $A_{j+1} \in \mathrm{maxenabled}(A_1 \dots A_j)$. Hence, $x = A_1 \dots A_{j+1}$ satisfies the conditions of (2). □

In the example at the beginning of Section 3, for instance, $x = bc$, $A_1 = \{b\}$, $A_2 = \{c\}$ satisfy the conditions of Theorem 6.1(2). We have here: $bc =^D \underline{\{b\}}\{c\}$, $A_1 = \{b\} \notin \mathrm{maxenabled}(\varepsilon) = \{\{a, b\}\}$, $A_2 \in \mathrm{maxenabled}(\underline{\{b\}}) = \mathrm{maxenabled}(b) = \{\{a\}, \{c\}\}$.

Theorem 6.1 characterises the inequality $\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$ by properties of some $x \in \mathrm{VFS}(P)$. We shall now formulate a stronger theorem which characterises this inequality by an element of $\mathrm{VMFS}(P)$. First, we may characterise the greatest VMFS-prefix.

Lemma 6.2. *Let* $x \in \mathrm{VFS}(P) - \mathrm{Pref}(\mathrm{VMFS}(P))$ *and* y *be the greatest prefix of* x *in* $\mathrm{VMFS}(P)$. *Then* $(\exists A_1, \dots, A_k \in \mathrm{Ind}(P))\ x = yA_1 \dots A_k \ \&\ A_1 \dots A_k =^D A_1 \dots A_k \ \&\ k \geqslant 2$.

Proof. Let $x =^D B_1 \dots B_m$. Suppose that $y \neq \varepsilon$ and $y =^D C_1 \dots C_p$. By Theorem 5.6 we have $p \geqslant m$ and $B_i = C_i$ for $i = 1, \dots, p$. Let us set $k = m - p$ (if $y = \varepsilon$, we set $k = m$ and $p = 0$) and $A_i = C_{p+i}$ for $i = 1, \dots, k$. Thus, $x = yA_1 \dots A_k$, $A_1 \dots A_k =^D A_1 \dots A_k$. Because $x \notin \mathrm{VMFS}(P)$, we have $k \geqslant 1$. Suppose $k = 1$. Then $x = yA_1$, $A_1 \in \mathrm{Ind}(P)$ and $y \in \mathrm{VMFS}(P)$. But this means that $A_1 \in \mathrm{enabled}(y) - \mathrm{maxenabled}(y)$, and there is a $B \in \mathrm{maxenabled}(y)$ such that $A_1 \subsetneq B$ and $yB \in \mathrm{VMFS}(P)$. Of course, yA_1 is a prefix of yB, so $x = yA_1 \in \mathrm{Pref}(\mathrm{VMFS}(P))$, a contradiction. □

Now, for the least VMFS-extensions, we get the following lemma.

Lemma 6.3. *Let* $x \in \mathrm{Pref}(\mathrm{VMFS}(P)) - \mathrm{VMFS}(P)$, y *is its greatest prefix in* $\mathrm{VMFS}(P)$, *and* y' *is one of its least extensions in* $\mathrm{VMFS}(P)$. *Then*

(1) $y' = yA_1 \dots A_k \ \&\ A_1 \dots A_k =^D A_1 \dots A_k \ \&\ k \geqslant 1$,
(2) $x = yB_1 \dots B_k \ \&\ B_1 \dots B_k =^D B_1 \dots B_k$,
(3) $y' = xC_1 \dots C_k$,
(4) $A_i = B_i \cup C_i$ *for* $i = 1, \dots, k$,
(5) $C_1 \neq \emptyset$,
(6) $(\forall i = 1, \dots, k)(\forall j = i, \dots, k)\ B_j \cup C_i \in \mathrm{Ind}(P)$.

Proof. Of course, there is a sequence $A_1, \dots, A_k$ such that $y' = yA_1 \dots, A_k$, $A_1 \in \mathrm{maxenabled}(y)$ and $A_i \in \mathrm{maxenabled}(yA_1 \dots A_{i-1})$ for $i = 2, \dots, k$. Note that $A_1, \dots, A_k$ satisfies (1) (see Theorem 5.4).

If $k = 1$, the lemma simply expresses that $y \neq x \neq y'$ and that we have yMy', $yCxCy'$.

For $k \geq 2$, let $x = yz_0$, $y' = xw$. For every $x \in \mathrm{Vev}(P)^*$, let $\mathrm{Ev}(x)$ denote the set of all events occurring in x. Let $B_1, \ldots, B_k$, $C_1, \ldots, C_k$ be the sets constructed by

(1) $B_1 = A_1 \cap \mathrm{Ev}(z_0)$, $C_1 = A_1 - B_1$,

(2) for $i = 1, \ldots, k-1$, z_i is constructed from z_{i-1} by dropping the first occurrence of each operation in B_i and $B_{i+1} = A_{i+1} \cap \mathrm{Ev}(z_i)$, $C_{i+1} = A_{i+1} - B_{i+1}$.

From the fact that $y' = yz_0w = xw = yA_1 \ldots A_k$ and the above construction, it follows: $A_i = B_i \cup C_i$, $x = yB_1 \ldots B_k$, $y' = xC_1 \ldots C_k$, i.e., the points (3), (4), and the first half of (2) are satisfied. Property (5) results from the fact that if $C_1 = \emptyset$, then $yB_1 = yA_1$ would be a greater VMFS prefix of x than y. Property (6) results from the fact that since $y' = yA_1 \ldots A_k = yB_1 \ldots B_kC_1 \ldots C_k$, each operation $c \in C_1$ has to commute with all the operations in $B_2, \ldots, B_k$; moreover, if $c = b \in B_j$, from the construction of B_1, c would belong to B_1 and not to C_1 since that means that $c \in \mathrm{Ev}(z_{j-1})$ and thus, $c \in \mathrm{Ev}(z_0)$; the same argument may then be resumed for $C_2, \ldots, C_k$ in that order. To prove that $B_1 \ldots B_k = {}^D B_1 \ldots B_k$ we have to show that

(i) $(\forall i = 1, \ldots, k)\ B_i \neq \emptyset$,

(ii) $(\forall b \in B_i)\ (b, B_{i-1}) \notin \mathrm{ind}$ for $i = 2, \ldots, k$.

Property (i) results from the facts that:

(1) if $B_k = \emptyset$, then $yA_1 \ldots A_{k-1}$ would be an extension of x in $\mathrm{VMFS}(P)$ smaller than y';

(2) let us suppose that i is the least index in $\{1, \ldots, k-1\}$ such that $B_i = \emptyset$; from (6) we know that each $c \in C_i = A_i$ is independent of each operation in $B_{i+1} (\neq \emptyset)$; but then A_i would not be maximally concurrent at $yA_1 \ldots A_{k-1}$ since we may add B_{i+1} to it.

Property (ii) results from an argument similar to the ones used above, namely, if $(\exists i \in \{2, \ldots, k\})(\exists b \in B_i)(\forall b' \in B_{i-1})\ (b, b') \in \mathrm{ind}$ (since from (6) $(\forall c \in C_{i-1})\ (b, c) \in \mathrm{ind}$), then $A_{i-1} = B_{i-1} \cup C_{i-1}$ would not be maximally concurrent at $yA_1 \ldots A_{i-2}$ since one may add b to it. □

Now, we are able to prove our main result.

Theorem 6.4. *The following properties are equivalent:*

(1) $\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$;

(2) *there are* $x \in \mathrm{VMFS}(P)$ *and nonempty event sets* $A_1, \ldots, A_k \in \mathrm{Ind}(P)$, *where* $k \geq 2$, *such that*

(a) $y = xA_1 \ldots A_k \in \mathrm{VFS}(P)$,

(b) $(\forall a \in \mathrm{Ev}(P))\ ya \in \mathrm{VFS}(P) \Rightarrow (a, A_{k-1} \cup A_k) \notin \mathrm{ind}$,

(c) *if* $z = A_1 \ldots A_k$, *then* $z = {}^D A_1 \ldots A_k$,

(d) $(\forall a \in A_k)(\exists b \in \mathrm{Ev}(P))\ xb \in \mathrm{VFS}(P)$ & $(b, A_1 \cup \cdots \cup A_{k-1} \cup (A_k - \{a\})) \in \mathrm{ind}$.

Proof. (1⇒2): For every $x \in \mathrm{VFS}(P)$, let $\mathrm{gp}(x)$ denote its greatest prefix in $\mathrm{VMFS}(P)$. We shall also denote $d(x) = m(x) - m(\mathrm{gp}(x))$ and $T = \mathrm{VFS}(P) - \mathrm{Pref}(\mathrm{VMFS}(P))$.

Let $V=\{x\in T\mid(\forall y\in T)\ d(x)\leqslant d(y)\}$. Note that $(\forall x,y\in V)\ d(x)=d(y)$. Let $k=d(x)$, where $x\in V$. From Lemma 6.2 we have $k\geqslant 2$. Let $x\in V$. We have $m=m(x)-k=m(\mathrm{gp}(x))\geqslant 0$ and x has the following il-decomposition: $x=^D B_1\dots B_m A_1\dots A_k$ (it may happen that $m=0$). Let $f,g: V\to\{0,1,2,\dots\}$ be functions defined as follows: if $x=^D B_1\dots B_m A_1\dots A_k$, then $f(x)=l(A_k)$, $g(x)=l(A_1\dots A_{k-1})$. Let us define $U=\{x\in V\mid(\forall y\in V)\ f(x)\leqslant f(y)\}$, $W=\{u\in U\mid(\forall y\in U)\ g(u)\geqslant g(y)\}$. We may notice that since $(\forall x\in V)\ g(x)\leqslant(k-1)\times\mathrm{card}(\mathrm{Ev}(P))$, W is a nonempty set (in fact, we have strict inequality, as A_1 may not be in maxenabled$(B_1\dots B_m)$). Since $y\in V$, $x=B_1\dots B_m=\mathrm{gp}(y)$ and $A_1\dots A_k=^D A_1\dots A_k$, i.e., (2)(a), (2)(c) are satisfied.

Suppose now that $ya\in\mathrm{VFS}(P)$ for some $a\in\mathrm{Ev}(P)$. Note that $ya\in T$ (since $ya\in\mathrm{Pref}(\mathrm{VMFS}(P))\Rightarrow y\in\mathrm{Pref}(\mathrm{VMFS}(P))$. By Theorem 5.3 we have $ya=^D B_1\dots B_m C_1\dots C_p$, where $p\geqslant k$, $A_i\subseteq C_i$ for $i=1,\dots,k$. Two cases may happen: $A_i=C_i$ for $i=1,\dots,k$ & $p=k+1$ & $C_p=\{a\}$; and: $(\exists j!)\ 1\leqslant j<k$ & $C_j=A_j\cup\{a\}$ & $C_i=A_i$ for $i\neq j$ & $k=p$.

Now, suppose that $(a, A_{k-1}\cup A_k)\in\mathrm{ind}$. In the first case, since $B_1,\dots,B_m$, $C_1,\dots,C_p$ is the il-decomposition, we have $(\exists b\in C_{p-1}=A_k)\ (a,b)\notin\mathrm{ind}$, a contradiction! Let us consider the second case. Note that $\mathrm{gp}(y)$ is a prefix of $\mathrm{gp}(ya)$, thus, $d(ya)\leqslant d(y)=k$, but this means that $ya\in V$. Next, $j<k$ and $f(ya)=f(y)$, so $ya\in U$. But $g(ya)=1+g(y)$, a contradiction since $y\in W$! Therefore, we have $(a, A_{k-1}\cup A_k)\notin\mathrm{ind}$, and condition (2)(b) is satisfied.

To prove (2)(d), let us suppose that $a\in A_k$ and $A=A_k-\{a\}$. We have to consider two cases.

Case 1: $A=\emptyset$. From the definition of V we have $z=xA_1\dots A_{k-1}\in\mathrm{Pref}(\mathrm{VMFS}(P))$.

Case 2: $A\neq\emptyset$. Suppose that $z=xA_1\dots A_{k-1}A\in T$. Note that $xA_1\dots A_{k-1}=^D B_1\dots B_m A_1\dots A_{k-1}$; hence, $z=^D B_1\dots B_m A_1\dots A_{k-1}A$. This means that $z\in V$, but $f(z)=f(y)-1$, a contradiction because $y\in U$! Thus, in both cases, $z=xA_1\dots A_{k-1}A\in\mathrm{Pref}(\mathrm{VMFS}(P))-\mathrm{VMFS}(P)$ (we set $\emptyset=\varepsilon$). By Lemma 6.3 we have $(\exists C_1\subseteq\mathrm{Ev}(P))\ C_1\neq\emptyset$, $zC_1\in\mathrm{VFS}(P)$, $A_1\cup C_1\in\mathrm{maxenabled}(x)$ & $(\forall b\in C_1)\ (b, A_1\cup\dots\cup A_{k-1}\cup A)\in\mathrm{ind}$, so that any $b\in C_1$ satisfies (2)(d).

(2$\Rightarrow$1): Let $x=^D B_1\dots B_m$. Then, from (2)(c) and from the fact that, by Theorem 5.5, $B_m\in\mathrm{maxenabled}(B_1\dots B_{m-1})$, we have $y=^D B_1\dots B_m A_1\dots A_k$. Since $A_k\neq\emptyset$, there is at least one $b\in\mathrm{Ev}(P)$ which satisfies (2)(d). Thus, $b\in\mathrm{enabled}(xA_1\dots A_{k-1})$ and $(b, A_{k-1})\in\mathrm{ind}$. Hence, $A_{k-1}\notin\mathrm{maxenabled}(xA_1\dots A_{k-2})$. Let A be such that $A\cup A_k\in\mathrm{maxenabled}(xA_1\dots A_{k-1})$. By (2)(b) we have

$$z=xA_1\dots A_{k-1}A_kA\ \Rightarrow\ z=^D B_1\dots B_m A_1\dots A_{k-1}\underline{A_k\cup A}.$$

Thus, by Theorem 6.1, $\mathrm{VFS}(P)\neq\mathrm{Pref}(\mathrm{VMFS}(P))$. □

The conditions of the previous theorem may, in fact, be strengthened. Indeed, we have the following lemma.

Lemma 6.5. *Let* $x \in \mathrm{VMFS}(P)$ *and let* $A_1, \ldots, A_k \in \mathrm{Ind}(P)$, *where* $k \geq 2$, *be those from Theorem* 6.4(2). *Then,*

(a) $A_1 \notin \mathrm{maxenabled}(x)$,

(b) $(\forall j < k)\ A_j \cap A_k = \emptyset$,

(c) $(\exists b \in \mathrm{Ev}(P))\ xA_1 \ldots A_{k-1}b \in \mathrm{VFS}(P)\ \&\ (b, A_{k-1}) \in \mathrm{ind}$.

Proof. (a) Let $b \in \mathrm{Ev}(P)$ satisfy Theorem 6.4(2)(d). Then $b \in \mathrm{enabled}(x)$ and $(b, A_1) \in \mathrm{ind}$, hence, $A_1 \notin \mathrm{maxenabled}(x)$.

(b) Let $a \in A_j \cap A_k$, where $j < k$. From Theorem 6.4(2)(d), it follows that $(\exists b \in \mathrm{Ev}(P))\ xb \in \mathrm{VFS}(P)\ \&\ (b, A_1 \cup \cdots \cup A_{k-1} \cup A) \in \mathrm{ind}$, where $A_k = A \cup \{a\}$. Since $a \in A_j$ and $j < k$, we have $(b, A_1 \cup \cdots \cup A_{k-1} \cup A_k) \in \mathrm{ind}$. But this means that $yb \in \mathrm{VFS}(P)$, and by Theorem 6.4(2)(b), $(b, A_{k-1} \cup A_k) \in \mathrm{ind}$, a contradiction.

(c) From Theorem 6.4(2)(d) we have $(\exists b \in \mathrm{Ev}(P))\ xb \in \mathrm{VFS}(P)\ \&\ (b, A_1 \cup \cdots \cup A_{k-1}) \in \mathrm{ind}$. But this means that $xA_1 \ldots A_{k-1}b \in \mathrm{VFS}(P)$ and $(b, A_{k-1}) \in \mathrm{ind}$. □

Consequently, we obtain the following theorem.

Theorem 6.6. *The following properties are equivalent:*

(1) $\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$;

(2) *there are* $x \in \mathrm{VMFS}(P)$ *and nonempty event sets* $A_1, \ldots, A_k \in \mathrm{Ind}(P)$, *where* $k \geq 2$, *such that*

(a) $y = xA_1 \ldots A_k \in \mathrm{VFS}(P)$,

(b) $(\forall a \in \mathrm{Ev}(P))\ ya \in \mathrm{VFS}(P) \Rightarrow (a, A_{k-1} \cup A_k) \notin \mathrm{ind}$,

(c) *if* $z = A_1 \ldots A_k$, *then* $z =^D A_1 \ldots A_k$,

(d) $(\exists b \in \mathrm{Ev}(P))\ xA_1 \ldots A_{k-1}b \in \mathrm{VFS}(P)\ \&\ (b, A_{k-1}) \in \mathrm{ind}$.

Proof. (1⇒2): This follows from Theorem 6.4 and Lemma 6.5(c).

(2⇒1): Let $x =^D B_1 \ldots B_m$. From (2)(c) and from the fact that, by Theorem 5.5, $B_m \in \mathrm{maxenabled}(B_1 \ldots B_{m-1})$, we have $y =^D B_1 \ldots B_m A_1 \ldots A_k$. By (2)(d) we have $A_{k-1} \notin \mathrm{maxenabled}(xA_1 \ldots A_{k-2})$. Let A be such that $A \cup A_k \in \mathrm{maxenabled}(xA_1 \ldots A_{k-1})$. By (2)(b) we have

$$z = xA_1 \ldots A_{k-1}A_kA \ \Rightarrow\ z =^D B_1 \ldots B_m A_1 \ldots A_{k-1}\underline{A_k \cup A}.$$

Thus, by Theorem 6.1, $\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$. □

In the case of the example at the beginning of Section 3, for example, $x = abd$, $A_1 = \{b\}$, $A_2 = \{c\}$ satisfy condition (2) of Theorems 6.4, 6.6, and Lemma 6.5. Here we have $y = abdbc \in \mathrm{VFS}(P)$ (condition (2)(a) of Theorems 6.4, 6.6); $A_1 \cap A_2 = \emptyset$ (Lemma 6.5(b)); $A_1 = \{b\} \notin \mathrm{maxenabled}(x) = \{\{a, b\}\}$ (Lemma 6.5(a)); $\mathrm{enabled}(y) = \{\{d\}\}$, thus,

$$ya \in \mathrm{VFS}(P)\ \&\ \alpha \in \mathrm{Ev}(P)\ \Rightarrow\ \alpha = d,$$

and $(d, \{b, c\}) \notin \text{ind}$ (condition (2)(b) of Theorems 6.4 and 6.6); $A_1A_2 = \boldsymbol{bc} = {}^D \underline{\{b\}\{c\}} = A_1A_2$ (condition (2)(c) of Theorems 6.4 and 6.6); $\boldsymbol{xa} \in \text{VFS}(P)$ and $(a, A_1 \cup (A_2 - \{c\})) = (a, b) \in \text{ind}$ (condition (2)(d) of Theorem 6.4); $\boldsymbol{xA_1a} = \boldsymbol{abdba} \in \text{VFS}(P)$ and $(a, A_1) = (a, b) \in \text{ind}$ (condition (2)(d) of Theorem 6.4 and Lemma 6.5(c)).

7. Simpler sufficient conditions

Theorems 6.1, 6.4, and 6.6 give us both necessary and sufficient conditions for the equivalence of the semantics 'execute as possible' and 'execute as much as possible in parallel'. In practice we are often satisfied by sufficient conditions for the equality $\text{VFS}(P) = \text{Pref}(\text{VMFS}(P))$ (i.e., necessary conditions for the inequality $\text{VFS}(P) \neq \text{Pref}(\text{VMFS}(P))$), but we need the conditions to be relatively easy to verify. In particular, it would be useful to exhibit structural criteria (i.e., depending on the static system description) instead of dynamic ones (i.e., depending on the existence of some type of history).

In this section we shall prove some nice, but rather wide (for practical applications), sufficient conditions.

Theorem 7.1. *If* $\text{VFS}(P) \neq \text{Pref}(\text{VMFS}(P))$, *then there are* $x \in \text{VFS}(P)$, $a, b, c \in \text{Ev}(P)$ *satisfying*

(1) $\boldsymbol{xab} \in \text{VFS}(P)$, $\boldsymbol{xac} \in \text{VFS}(P)$;

(2) $a \neq b$, $(a, b) \notin \text{ind}$, $(b, c) \notin \text{ind}$, $(a, c) \in \text{ind}$.

Proof. Let us take x, $A_1, \ldots, A_k$ from the proof of Theorem 6.4. Since $A_1 \ldots A_k = {}^D A_1 \ldots A_k$, if we take $a \in A_{k-1}$ and $b \in A_k$ (from Lemma 6.5(b) it follows that $A_{k-1} \cap A_k = \emptyset$), we have $(a, b) \notin \text{ind}$. From the proof of Theorem 6.4, we also have $xA_1 \ldots A_{k-1}\underline{A_k - \{b\}} \in \text{Pref}(\text{VMFS}(P))$ (provided $\boldsymbol{\emptyset} = \boldsymbol{\varepsilon}$), so that, by Lemma 6.3, there exists a $C \subseteq \text{Ev}(P)$ such that $C \neq \emptyset$, $xA_1 \ldots A_{k-1}\underline{A_k - \{b\}}C \in \text{VFS}(P)$, $A_1 \cup C \in \text{maxenabled}(x)$, $(\forall c \in C)$ $(c, A_k - \{b\}) \in \text{ind}$, and $(\forall i = 1, \ldots, k-1)$ $(\forall c \in C)$ $(c, A_i) \in \text{ind}$. Let $c \in C$. Suppose that $(b, c) \in \text{ind}$. In such a case $(c, A_{k-1} \cup A_k) \in \text{ind}$ and $xA_1 \ldots A_{k-1}c \in \text{VFS}(P)$, which contradicts Theorem 6.4(2)(b). Thus, $(b, c) \notin \text{ind}$. Of course, $(a, c) \in \text{ind}$ since $a \in A_{k-1}$. Let $z = xA_1 \ldots A_{k-2}\underline{A_{k-1} - \{a\}}$; we have $\boldsymbol{zab} \in \text{VFS}(P)$ and $\boldsymbol{zac} \in \text{VFS}(P)$. □

It may be noticed that the conditions of Theorem 7.1 correspond, in fact, in the context of Petri nets, to a situation of asymmetric confusion (see [3, p. 526]) and confusion is known to be connected to awkward problems (see [30]).

Various conditions formulated in [6, 12, 17] are now simple consequences of Theorem 7.1.

Corollary 7.2 ([6, 12])

$$\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$$
$$\Rightarrow (\exists x \in \mathrm{VFS}(P))(\exists a, b, c \in \mathrm{Ev}(P))\ \boldsymbol{xab} \in \mathrm{VFS}(P)\ \&\ \boldsymbol{xac} \in \mathrm{VFS}(P)\ \&\ a \neq b \&\ b \neq c\ \&\ (a, c) \in \mathrm{ind}\ \&\ (b, c) \notin \mathrm{ind}.$$

Proof. Let x, a, b, c be as in Theorem 7.1. Note that $(a, b) \notin \mathrm{ind}$ and $(a, c) \in \mathrm{ind}$ imply $b \neq c$. □

Corollary 7.3 ([17]).[2] *If* $\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$, *then there are* $a, b, c \in \mathrm{Ev}(P)$ *satisfying*:

(1) $a \neq b\ \&\ (\exists i)(\exists x \in \mathrm{FS}(P_i))\ xab \in \mathrm{FS}(P_i)$,

(2) $b \neq c\ \&\ (\exists j)(\exists y \in \mathrm{FS}(P_j))\ yb \in \mathrm{FS}(P_j)\ \&\ yc \in \mathrm{FS}(P_j)$,

(3) $(a, c) \in \mathrm{ind}\ \&\ (\exists z \in \mathrm{VFS}(P))\ \boldsymbol{zac} \in \mathrm{VFS}(P)$.

Proof. The triple a, b, c from Theorem 7.1 satisfies (1), (2), (3). □

In the case of the example at the beginning of Section 3 ε, b, c, a (in this order) satisfy the right-hand side of the implication in Theorem 7.1 (and Corollaries 7.2, 7.3, as well). The reverse of Theorem 7.1 (and Corollaries 7.2, 7.3) is not true. To prove this, let us consider the following example:

$P =$ **path** $a; b; d$ **end**
path $c, (b; d)$ **end**
path $c, e; d$ **end.**

In this case, we have: $\mathrm{VMFS}(P) = (\boldsymbol{acbd} \cup \boldsymbol{aebd})^*$, $\mathrm{VFS}(P) = (\boldsymbol{acbd} \cup \boldsymbol{aebd})^* \{\varepsilon \cup \boldsymbol{a} \cup \boldsymbol{ab} \cup \boldsymbol{aeb} \cup \boldsymbol{c} \cup \boldsymbol{ac} \cup \boldsymbol{acb} \cup \boldsymbol{e} \cup \boldsymbol{ae}\}$, so $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$, but $\varepsilon \in \mathrm{VMFS}(P)$, $\boldsymbol{\varepsilon ab} \in \mathrm{VFS}(P)$, $\boldsymbol{\varepsilon ac} \in \mathrm{VFS}(P)$, $(a, b) \notin \mathrm{ind}$, $(b, c) \notin \mathrm{ind}$, $(a, c) \in \mathrm{ind}$, thus, ε, a, b, c satisfy the right-hand side of the implication in Theorem 7.1.

From Theorem 7.1 it follows that if $\mathrm{VFS}(P) \neq \mathrm{Pref}(\mathrm{VMFS}(P))$, then $\mathrm{Ev}(P)$ consists of three elements at least. The simple example at the beginning of Section 3 has $\mathrm{Ev}(P) = \{a, b, c, d\}$ and is not minimal. Here is probably the simplest one ($\mathrm{Ev}(P) = \{a, b, c\}$):

$P =$ **path** a, b **end**
path c, b **end.**

Note that here, $\boldsymbol{cb} \in \mathrm{VFS}(P) - \mathrm{Pref}(\mathrm{VMFS}(P))$.

Let us now define the following relations on $\mathrm{Ev}(P) \times \mathrm{Ev}(P)$:

- $(a, b) \in \mathrm{pre} :\Leftrightarrow (\exists i)(\exists x \in \mathrm{FS}(P_i))\ xab \in \mathrm{FS}(P_i)\ \&\ a \neq b$;
- $(a, b) \in \mathbf{pre} :\Leftrightarrow a \neq b\ \&\ (a, b) \notin \mathrm{ind}\ \&\ (\exists x \in \mathrm{VFS}(P))\ \boldsymbol{xab} \in \mathrm{VFS}(P)$;
- $(a, b) \in \mathrm{prel} :\Leftrightarrow a \neq b\ \&\ (a, b) \notin \mathrm{ind}\ \&\ ((\forall i \in r(a) \cap r(b))(\exists x \in \mathrm{FS}(P_i))\ xab \in \mathrm{FS}(P_i))$;
- $(a, b) \in \mathrm{exc} :\Leftrightarrow a \neq b\ \&\ ((\exists i)(\exists x \in \mathrm{FS}(P_i)\ xa \in \mathrm{FS}(P_i)\ \&\ xb \in \mathrm{FS}(P_i))$;

[2] The proof in [17] of this corollary contains a serious error.

- $(a, b) \in \mathbf{exc} :\Leftrightarrow a \neq b \;\&\; (a, b) \notin \mathrm{ind} \;\&\; ((\exists x \in \mathrm{VFS}(P))\; \boldsymbol{xa} \in \mathrm{VFS}(P) \;\&\; \boldsymbol{xb} \in \mathrm{VFS}(P))$;
- $(a, b) \in \mathrm{excl} :\Leftrightarrow a \neq b \;\&\; (a, b) \notin \mathrm{ind} \;\&\; ((\forall i \in r(a) \cap r(b))(\forall x \in \mathrm{FS}(P_i))\; xa \in \mathrm{FS}(P_i) \;\&\; xb \in \mathrm{FS}(P_i))$;
- $(a, b) \in \mathrm{con} :\Leftrightarrow (a, b) \in \mathrm{ind} \;\&\; (\forall x \in \mathrm{VFS}(P))\; \boldsymbol{xab} \in \mathrm{VFS}(P)$.

In terms of Petri nets associated to P, the relations pre, **pre**, exc, **exc** and con describe the simple situations shown in Fig. 4, where m denotes a marking reachable from the initial marking and dashed lines denote context. Note that there are cases where $(a, b) \in \mathrm{pre} \cap \mathrm{exc}$ (see the last example)!

Corollary 7.4

(1) $\mathbf{pre} \subseteq \mathrm{prel} \subseteq \mathrm{pre}$.
(2) $\mathbf{exc} \subseteq \mathrm{excl} \subseteq \mathrm{exc}$.
(3) $\mathrm{con} \subseteq \mathrm{ind}$.

Note that the relations: pre, prel, exc, excl, and ind can be described by analysing $P = P_1 \ldots P_n$ on *the syntax level only* (i.e., by analysing expressions $P_1, \ldots, P_n$ treated as strings of symbols or by analysing the graph of a Petri net associated to P).

Let $\mathrm{PDT}_0(P)$, $\mathrm{PDT}_1(P)$, $\mathrm{PDT}_2(P)$, $\mathrm{PDT}_3(P) \subseteq \mathrm{Ev}(P) \times \mathrm{Ev}(P) \times \mathrm{Ev}(P)$ be the following relations: [3]

- $(a, b, c) \in \mathrm{PDT}_0(P) :\Leftrightarrow (a, b) \in \mathrm{pre} \;\&\; (b, c) \in \mathrm{exc} \;\&\; (a, c) \in \mathrm{ind}$;
- $(a, b, c) \in \mathrm{PDT}_1(P) :\Leftrightarrow (a, b) \in \mathrm{pre} \;\&\; (b, c) \in \mathrm{exc} \;\&\; (a, c) \in \mathrm{con}$;
- $(a, b, c) \in \mathrm{PDT}_2(P) :\Leftrightarrow (a, b) \in \mathbf{pre} \;\&\; (b, c) \in \mathbf{exc} \;\&\; (a, c) \in \mathrm{con}$;
- $(a, b, c) \in \mathrm{PDT}_3(P) :\Leftrightarrow (a, b) \in \mathrm{prel} \;\&\; (b, c) \in \mathrm{excl} \;\&\; (a, c) \in \mathrm{ind}$.

The name PDT is an abbreviation of Potentially Dangerous Triple.

Theorem 7.5

$$\mathrm{PDT}_0(P) = \emptyset \begin{array}{c} \nearrow \mathrm{PDT}_1(P) = \emptyset \searrow \\ \searrow \mathrm{PDT}_3(P) = \emptyset \nearrow \end{array} \mathrm{PDT}_2(P) = \emptyset \Rightarrow \mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P)).$$

Proof. The first four implications follows from Corollary 7.4. The last implication is equivalent to Theorem 7.1, with the additional requirement that the same x may be used for **pre** and con, and that xa may be used for x in **exc**. □

One may show that, in general, no implication from Theorem 7.5 may be replaced by equivalence (although it may happen that for some subclasses we have some equivalences). In particular, in the case of the example after Corollary 7.3, we have $(a, b, c) \in \mathrm{PDT}_i(P)$, $i = 0, 1, 2, 3$ and $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$. Note that for $i = 0$ and $i = 3$ the verification of equality $\mathrm{PDT}_i(P) = \emptyset$ can be done on the *syntax level only*. Experiences of the first two authors show that the conditions $\mathrm{PDT}_0(P) = \emptyset$,

[3] Because we now have stronger results, these PDT_i, for $i = 0, 1, 2, 3$ differ slightly from the similar concepts of [6, 12]. PDT_1 defined above is equivalent to PDT of [17].

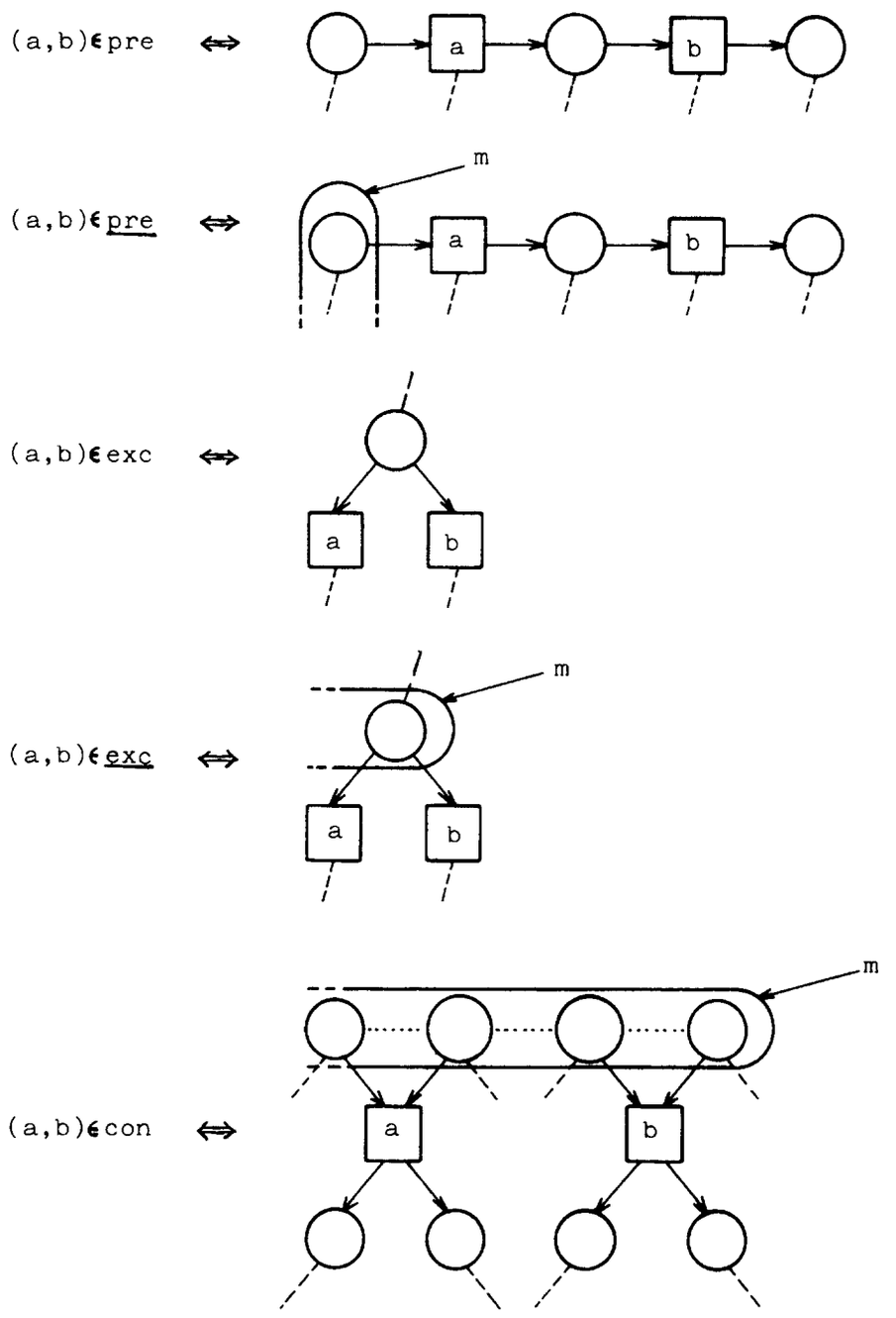


Fig. 4.

$PDT_1(P) = \emptyset$, $PDT_3(P) = \emptyset$ are sufficiently weak to have wide applications (for instance, they may be applied to all examples from [15]).

It turns out that some special forms of the relations exc, excl, exc also guarantee the equality $VFS(P) = Pref(VMFS(P))$.

Theorem 7.6

(1) $[(\forall a, b \in Ev(P))\,(a, b) \in exc \Rightarrow r(a) = r(b)] \Rightarrow PDT_0(P) = \emptyset.$

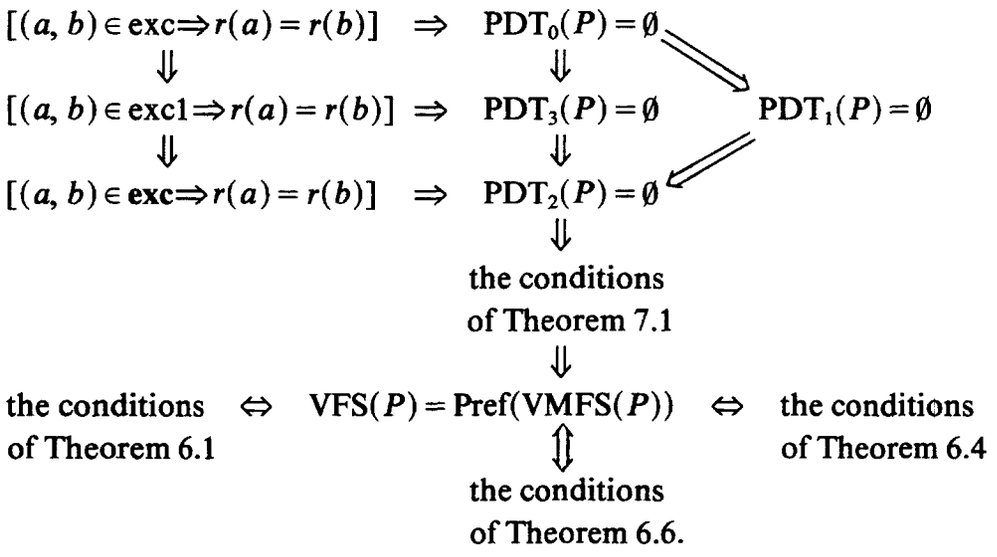
(2) $[(\forall a, b \in Ev(P))\,(a, b) \in excl \Rightarrow r(a) = r(b)] \Rightarrow PDT_3(P) = \emptyset.$

(3) $[(\forall a, b \in \mathrm{Ev}(P))\ (a, b) \in \mathbf{exc} \Rightarrow r(a) = r(b)] \Rightarrow \mathrm{PDT}_2(P) = \emptyset$.

Proof. (1) Suppose $(\forall a, b \in \mathrm{Ev}(P))\ (a, b) \in \mathrm{exc} \Rightarrow r(a) = r(b)$ and $(a, b, c) \in \mathrm{PDT}_0(P)$. This means that $(a, b) \in \mathrm{pre}$, $(b, c) \in \mathrm{exc}$, $(a, c) \in \mathrm{ind}$. But $(a, b) \in \mathrm{pre} \Rightarrow r(a) \cap r(b) \neq \emptyset$, $(a, c) \in \mathrm{ind} \Rightarrow r(a) \cap r(c) = \emptyset$. On the other hand, $(b, c) \in \mathrm{exc} \Rightarrow r(c) = r(b)$, a contradiction. For (2) and (3), we proceed similarly. □

The full relationships among various conditions for the equality $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$ are given by the corollary below.

Corollary 7.7



8. Conclusion

The results presented here enlighten somewhat the mixed structure of VFS and VMFS and give some necessary and sufficient conditions for their equivalence. Those conditions may be interpreted as an answer to the question: "When are the two semantics of concurrent systems: 'execute as possible' and 'execute as much as possible in parallel' equivalent?".

Many problems are left open, however. Some of them are:

- Find fast algorithms to check if a couple of operations belongs to **pre**, pre, pre1, **exc**, exc, excl, or con.
- Is it possible to refine the conditions in Theorems 6.1, 6.4, 6.6 and 7.1?
- How large is the class of generalised paths where $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VMFS}(P))$?
- If $x \in \mathrm{VFS}(P) - \mathrm{Pref}(\mathrm{VMFS}(P))$, we may define $\mathrm{VMFS}(P, x) = \{y \mid xM^*y\}$; what type of results may be obtained from there?

All results of the paper may easily be translated into a language of labelled Petri nets (instead of vector firing sequences, we can use Mazurkiewicz traces [21, 22], occurrence nets [3, p. 527], or slightly modified subset languages [26]).

Acknowledgment

This work was begun when the authors (except M. Koutny) were in the Computing Laboratory, University of Newcastle upon Tyne, United Kingdom. John Cotronis contributed to the initial formulation of the problem considered. R. Janicki owes a debt to Povl Villumsen who organised his visit in Aalborg, which made it possible to finish the paper. Finally, the authors would like to thank the anonymous referees for his (or her) helpful comments.

References

[1] E. Best, Adequacy properties of path programs, *Theoret. Comput. Sci.* **18** (1982) 149-171.
[2] M. Blanchard et al., Le Grafcet, pour une représentation normalisée du cahier des charges d'un automatisme logique, *Automatique et Informatique Industrielle* **41–42** (1977).
[3] W. Brauer, ed., *Applications and Theory of Petri Nets*, Lecture Notes in Computer Science **84** (Springer, Berlin, 1980).
[4] W. Brauer, How to play the token game or difficulties in interpreting Place/Transition nets, *Petri Nets and Related System Models, Newsletter* **16** (1984) 3-13.
[5] H.D. Burkhard, Ordered firing in Petri nets, *Elektron. Informationsverarb. Kybernet.* **17**(2, 3) (1981) 71-86.
[6] R. Devillers, On the maximally concurrent evolution of a COSY system, Rept. ASM/106, Computing Laboratory, University of Newcastle upon Tyne, 1983.
[7] P. Enjalbert and M. Michel, Many-sorted temporal logic for multiprocesses systems, *Lecture Notes in Computer Science* **176** (1984) 273-281.
[8] B.C. Hamshare, A computer based environment for the design and analysis of concurrent systems: SIMULA implementation of the COSY notation, *Proc. of the 11th Ann. Conf. of the Association of Simula Users*, Paris 1983.
[9] C.A.R. Hoare, Communicating Sequential Processes, in: R.H. McKeag and A.M. Macnaghten, eds., *On the Construction of Programs* (Cambridge University Press, London/New York, 1980) 229-254.
[10] R. Janicki, An equivalence notion for path expression systems, *Elektron. Informationsverarb. Kybernet.* **21**(6) (1985) 283-285.
[11] R. Janicki, Transforming sequential systems into concurrent systems, *Theoret. Comput. Sci.* **36** (1985) 27-58.
[12] R. Janicki, P.E. Lauer and R. Devillers, Maximally concurrent evolution of non-sequential systems, *Proc. 4th European Workshop on Applications and Theory of Petri Nets*, Toulouse 1983, 188-202.
[13] K. Jensen et al., Petri net package: User's manual, Rept. DAIMI MD-46, Dept. of Computer Science, Aarhus University, 1983.
[14] E. Knuth, Petri nets and trace languages, *Proc. 1st European Conf. on Parallel and Distributed Processing*, Paris 1979, 51-56.
[15] P.E. Lauer, Computer system dossiers, in: Y. Parker and J.P. Verjus, eds., *Distributed Computing Systems: Synchronisation, Control and Communication* (Academic Press, New York/London, 1983) 109-148.
[16] P.E. Lauer, User's introduction to BCS, Rept. ASM/107, Computing Laboratory, University of Newcastle upon Tyne, 1983.

[17] P.E. Lauer and R. Janicki, The role of maximally concurrent simulation in the computer based analysis of distributed systems, Rept. ASM/96, Computing Laboratory, University of Newcastle upon Tyne, 1982.

[18] P.E. Lauer, M.W. Shields and E. Best, Formal theory of the basic COSY notation, Tech. Rept. TR-143, Computing Laboratory, University of Newcastle upon Tyne, 1979.

[19] P.E. Lauer, M.W. Shields and J.Y. Cotronis, Formal behavioural specifications of concurrent systems without globality assumptions, *Lecture Notes in Computer Science* **107** (Springer, Berlin, 1981) 115–151.

[20] P.E. Lauer, P.R. Torrigiani and M.W. Shields, COSY: A system specification language based on path expressions, *Acta Inform.* **12** (1979) 109–158.

[21] A. Mazurkiewicz, Concurrent Program Schemes and Their Interpretations, Rept. DAIMI PB-78, Dept. of Computer Science, Aarhus University, 1977.

[22] A. Mazurkiewicz, Traces, histories, graphs: Instances of a process monoid, *Lecture Notes in Computer Science* **176** (Springer, Berlin, 1984) 115–133.

[23] R. Milner, *A Calculus for Communicating Systems*, Lecture Notes in Computer Science **92** (Springer, Berlin, 1980).

[24] B. Montel et al., OVIDE, a software package for the validation of systems represented by Petri Net based models, *Proc. 4th European Workshop on Application and Theory of Petri Nets*, Toulouse 1983, 229–308.

[25] J.L. Peterson, Petri nets, *ACM Comput. Surveys* **9**(3) (1977) 223–252.

[26] G. Rozenberg and R. Verraedt, Subset languages of Petri nets, *Informatik-Fachberichte* **66** (Springer, Berlin, 1983) 250–263.

[27] A. Salwicki and T. Müldner, On algorithmic properties of concurrent programs, *Lecture Notes in Computer Science* **125** (Springer, Berlin, 1981) 169–197.

[28] M.W. Shields, Adequate path expressions, *Lecture Notes in Computer Science* **70** (Springer, Berlin, 1979) 249–265.

[29] M.W. Shields, Non-sequential behaviour 1, Internal Rept. CSR-120-82, Dept. of Computer Science, University of Edinburgh, 1982; short version in: *Lecture Notes in Computer Science* **167** (Springer, Berlin, 1984) 229–239.

[30] M.W. Shields, On the non-sequential behaviour of systems possessing a generalised free-choice property, Internal Rept. CSR-92-81, Dept. of Computer Science, University of Edinburgh, 1981.

[31] E. Szpilrajn, Sur l'extension de l'ordre partial, *Fund. Math.* **16** (1930) 386–389.