TRANSFORMING SEQUENTIAL SYSTEMS INTO CONCURRENT SYSTEMS

Ryszard JANICKI*

Institute of Electronic Systems, Aalborg University Centre, Strandvejen 19, DK-9000 Aalborg, Denmark

Communicated by R. Milner
Received September 1982
Revised May 1984

Abstract. A problem of concurrent system specification is studied. A functionally equivalent system is first specified, then a set of independent actions or abstract resources is devised, and, finally, this sequential system is transformed into an equivalent concurrent system. The method is based on the theory of path expressions. The notion of functional equivalence is formally defined and studied. Necessary and sufficient conditions, stating when the method can be used, are formulated and proved. Some examples (vending machine, cigarette smokers, readers and writers, dining philosophers) are discussed.

Introduction

Concurrent systems are more difficult to design and analyse than sequential ones because they can exhibit extremely complicated behaviour. Furthermore, it is very difficult to comprehend the total effect of actions being performed concurrently and with independent speeds. In practice, when a problem is complicated itself, the first solution is frequently sequential, and only later solutions are concurrent. This is almost a standard procedure in the case of technological processes.

In [6, 7], a method for developing a concurrent system from a functionally equivalent sequential system was suggested.

In this paper we extend the ideas of [6, 7] and apply them to the COSY Formalism proposed by Peter Lauer's group [10, 11, 12, 13, 14, 20, 23].

The method consists in starting with the sequential system, determining a set of independent actions (by means of so-called abstract resources), and then performing a set of transformations of the sequential system resulting in a concurrent system.

The notion of functional equivalence is formally defined and suitable necessary and sufficient conditions are formulated and proved. Some new concepts of the COSY Vector Firing Sequence Semantics are also presented.

* The main part of this work was carried out during the author's visit at the Computing Laboratory of the University of Newcastle-upon-Tyne. Some part was also done at the Institute of Mathematics of the Warsaw Technical University. The author is on leave from Warsaw Technical University.

The following well-known examples are discussed: a noisy vending machine [4, 10], cigarette smokers [19], readers and writers [1], and dining philosophers [2].

The approach presented in this paper follows from the author's conviction that our mental perception of reality is sequential (see [9]), thus, in many cases starting with a sequential version is the easiest way of designing a concurrent system. The similar viewpoint (but the different level of abstraction) is presented in [16, 17], where concurrent systems are specified in two steps: first, a correct program that can be implemented sequentially is refined, and next, so-called semantics relations allowing relaxations in the sequencing of the refinements operations (e.g., concurrency) are defined.

For those who are not convinced that this is a useful way of constructing systems, or indeed, an advisable way of thinking about them, or who like purely theoreoretical formulations, the problem considered in this paper may be formulated as follows. We are given a system described by a regular expression with an outermost Kleene star; the alphabetical symbols represent possible actions of the system and the regular language associated with the regular expression determines the set of legitimate sequences of occurrence of these actions. We are also given a collection of 'abstract resources'. An abstract resource is associated to a set of action names; the resource may only be accessed by these actions associated and they must be performed in sequence. Together, the expression and the collection of resources determine a language of objects (actually, vectors of strings) which describe all possible concurrent behaviours involving these actions, which, first, are such that some sequentialization of the behaviour is a sequence belonging to the regular language, second, two actions are only sequenced if they access a common abstract resource. The problem is to construct a path expression accepting the asynchronous language. This is a particular case of a general problem to find conditions under which an asynchronous language is a 'product' of string languages.

All results of this paper can easily be translated into the formalism of labelled Petri nets and Mazurkiewicz traces (see [13, 14, 18, 22]).

In Section 1, a brief description of the COSY Formalism is presented. Section 2 contains the method description. The notion of functional equivalence is formally defined in Section 3. Necessary and sufficient conditions for the functional equivalence and an algorithm for the verification are presented in Section 4. Section 5 is devoted to applications of the method. In Section 6, the longest in this paper, a proof of necessary and sufficient conditions is presented. Section 7 contains a final comment.

Some results of this paper have already been published (see [9]).

1. A brief description of COSY

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

A basic COSY program or generalized path is a collection of single paths enclosed in **system** and **endsystem** parenthesis.

A single path is a regular expression enclosed by **path** and **end**. For example:

```
P = system
        path a ; b, c end
        path (d ; e)* ; b end
    endsystem
```

In every regular expression like the above, the semicolon implies sequence (concatenation), and the comma implies mutually exclusive choice. The comma binds more strongly than semicolon, so that the sequence a ; b, c means "first a, then either b or c". A sequence may be enclosed in conventional parentheses with a Kleene star appended, as for instance $(d\ ;\ e)^*$, which means that the enclosed sequence may be executed zero or more times. The sequence appearing between **path** and **end** is implicitly so enclosed, so that paths describe cyclic sequences of actions. The synchronization among paths is due to common actions ("b" in the above example). Every single path describes a sequential system or subsystem.

For more details, the reader is referred to [10, 11, 12, 13, 14, 15].

2. Definition of the method

First, we will explain the method by analysing a very simple example: Hoare's noisy vending machine (see [4, 10]), and next we will formally define the method.

Consider a vending machine which may be used by two customers concurrently, that is, a machine that has distinct slots for 5 penny and 10 penny coins, and two distinct points for extraction of small and large packets of biscuits.

This machine may involve the following actions:

5p—insertion of a 5 penny coin,
10p—insertion of a 10 penny coin,
small—withdrawal of a small packet of biscuits,
large—withdrawal of a large packet of biscuits,
plunk—sound made by a small packet of biscuits dropping out of the machine,
plonk—sound made by a large packet of biscuits dropping out of the machine.

The system described above is very simple and it can easily be specified by a generalized path (see [10]), but we assume that we do not know how to specify this system concurrently, while we are able to specify it sequentially.

The single path specifying the sequential vending machine (at any moment only one customer uses a machine) is of the following form:

P_S = **path** (5p ; small ; plunk), (10p ; large ; plonk) **end**.

This sequential solution is not the only one and not even the most general, but it seems to define quite precisely a *function* of this system. The function of a vending machine is to vend biscuits. All what P_S does is to perform certain actions in sequence, but sequences 5p.small.plunk and 10p.large.plonk may be interpreted as events: selling one small packet of biscuits, and selling one large packet of biscuits. In this sense, P_S may be treated as a description of the function of our system.

The full specification of every system consists, in fact, of two parts *at least*. The first part described a *function* of a system, that is, it defines what the system does; whereas the second part describes *resources* necessary to perform the function of a system.

In the case of a vending machine system we can distinguish four resources:
SVM—a part of the machine, which vends small packets of biscuits,
LVM—a part of the machine, which vends large packets of biscuits,
SC—a customer asking for a small packet of biscuits, and
LC—a customer asking for a large packet of biscuits.

Let r denote the function describing which resources are necessary to perform each action, and $\hat{r}$ denote the function describing which actions are associated to each resource. Thus we have:

$$r(\text{5p}) = \{\text{SVM}, \text{SC}\},$$
$$r(\text{10p}) = \{\text{LVM}, \text{LC}\},$$
$$r(\text{small}) = \{\text{SVM}, \text{SC}\},$$
$$r(\text{large}) = \{\text{LVM}, \text{LC}\},$$
$$r(\text{plunk}) = \{\text{SVM}\},$$
$$r(\text{plonk}) = \{\text{LVM}\},$$
$$\hat{r}(\text{SVM}) = \{\text{5p}, \text{small}, \text{plunk}\},$$
$$\hat{r}(\text{LVM}) = \{\text{10p}, \text{large}, \text{plonk}\},$$
$$\hat{r}(\text{SC}) = \{\text{5p}, \text{small}\},$$
$$\hat{r}(\text{LC}) = \{\text{10p}, \text{large}\}.$$

Note that $\hat{r}$ is fully described by r, namely, for every resource x, $\hat{r}(x) = \{a \mid x \in r(a)\}$.

If we assume that actions may be performed concurrently only if they use no common resource, then for every resource x the set $\hat{r}(x)$ contains all actions that must be performed only one at a time.

The next step of our method is the *projection on resources*. Let us consider the resource SVM. We have $\hat{r}(\text{SVM}) = \{\text{5p}, \text{small}, \text{plunk}\}$. At first we replace in P_S all actions except 5p, small, plunk by the symbol "ε" (empty string).

As the result we obtain the path

path (5p ; small ; plunk), (ε ; ε ; ε) **end**.

Next we replace the above path by an equivalent one, in the sense of generating the same regular language, but without the symbol "ε".

This new path is now of the form

path 5p ; small ; plunk **end**

and it will be denoted by the symbol P_S/SVM.

In a similar way we can obtain the following paths:

P_S/LVM : **path** 10p ; large ; plonk **end**
P_S/SC : **path** 5p ; small **end**
P_S/LC : **path** 10p ; large **end**

The generalized path

P_C = **system**
P_S/SVM : **path** 5p ; small ; plunk **end**
P_S/LVM : **path** 10p ; large ; plonk **end**
P_S/SC : **path** 5p ; small **end**
P_S/LC : **path** 10p ; large **end**
endsystem

describes our final concurrent solution. Note that the identical P_C was also derived by Lauer [10] by informal arguments.

It seems to be intuitivety obvious that in the case of the vending machine system, the sequential single path P_S and the interconnected generalized path P_C are 'functionally equivalent', although this notion should be precisely defined and explained. This will be done in the next section.

We will now proceed with the *formal definition of our method.* Let P_S = **path** body **end** be any single path, and let Alpha(P_S) denote the set of all action names appearing in P_S. The path P_S is interpreted as a *sequential solution.*

Let resource(P_S) be any finite set (satisfying: resource(P_S) $\cap$ Alpha(P_S) = $\emptyset$) which is interpreted as the set of all *abstract resources associated with* P_S.

Let r: Alpha(P_S) $\rightarrow 2^{\mathrm{resource}(P_S)}$ be any total function. The function r will be called a *resource association function.*

Let $\hat{r}$: resource(P_S) $\rightarrow 2^{\mathrm{Alpha}(P_S)}$ be a function defined by

$$(\forall x \in \mathrm{resource}(P_S)) \quad \hat{r}(x) = \{a \mid x \in r(a)\}.$$

The function $\hat{r}$ describes which actions are associated to each resource and it will be called an *action distribution function.*

Let $x \in$ resource(P_S). By a *projection of* P_S *on* x, denoted by P_S/x, we mean any path derived from P_S in the following two steps:

(1) Every action symbol $a \in$ Alpha(P_S) $-\ \hat{r}(x)$ is replaced by the symbol "ε" (empty string). Assume that a new path obtained after this step is of the form **path** $\mathrm{body}_x^\varepsilon$ **end**.

(2) The regular expression $\mathrm{body}_x^\varepsilon$ is replaced by any ε-free regular expression body_x such that

$$|\mathrm{body}_x^\varepsilon| - \{\varepsilon\} = |\mathrm{body}_x|,$$

where $|\text{body}_x^\varepsilon|$ and $|\text{body}_x|$ denote languages defined by appropriate expressions (an algorithm may be found, for instance, in [3]).

In other words, P_S/x is derived from P_S by 'erasing' all symbols except those from $\hat{r}(x)$.

Assume that $\text{resource}(P_S)=\{x_1,\ldots,x_n\}$.

A generalized path P_C of the form

$$P_C=P_S/x_1\ldots P_S/x_n$$

is said to be *derived from* P_S *and* r.

One can easily prove that, for every single path P_S and every resource association function r, a generalized path P_C is always correctly defined. Unfortunately, it turns out that sometimes P_S and P_C are 'functionally different'. Conditions describing when they are 'functionally equivalent' will be discussed in detail in Section 4.

We assume that actions may be performed concurrently only if they use no common resources, i.e., the *independence relation* $I\subseteq \text{Alpha}(P_S)\times\text{Alpha}(P_S)$ is defined by the following equivalence:

$$(\forall a,b\in\text{Alpha}(P_S))\quad (a,b)\in I\;\Leftrightarrow\; r(a)\cap r(b)=\emptyset.$$

Thus the set $\hat{r}(x)$ contains all actions that must be performed only one at a time, and the relations I fulfills the following equivalence:

$$(a,b)\in I\;\Leftrightarrow\;[(a\neq b)\;\&\;(\forall x\in\text{resource}(P_S))\;a\notin\hat{r}(x)\text{ or }b\notin\hat{r}(x)],$$

so, using the terminology of [5, 8], it can be treated as a symmetric and irreflexive relation defined by the covering $\text{cov}=\{\hat{r}(x)\,|\,x\in\text{resource}(P_S)\}$ (such a relation R is defined by a covering cov iff $(a,b)\in R\Leftrightarrow a\neq b\;\&\;(\forall A\in\text{cov})\;a\notin A$ or $b\notin A$).

In the example considered above, the set $\text{resource}(P_S)$ is identical with the set of real physical resources of a system, but such a situation *is not a rule.* Following [5] we call the set $\text{resource}(P_S)$ the set of *abstract resources*; an abstract resource may be associated with a set of actions which, *for reasons of data protection or others,* must be performed only one at a time. It was proved in [5] that every symmetric and irreflexive relation can be defined by means of a set of abstract resources and a resource association function. Shields [21] has proposed the name 'abstract monitors' for sets $\hat{r}(x)$, where $x\in\text{resource}(P_S)$.

Sometimes, the independence relation I alone is much easier to define than the set $\text{resource}(P_S)$ and the function r (see Section 5.2). In such a case we may construct the set $\text{resource}(P_S)$ and the function r on the basis of I. The procedure is the following (see [5]). Let $I\subseteq\text{Alpha}(P_S)\times\text{Alpha}(P_S)$ be any symmetrical and irreflexive relation (interpreted as an *independence relation*).

Let $\overline{\text{kens}}(I)\subseteq 2^{\text{Alpha}(P_S)}$ be the following family of sets (see [5, 6, 7, 8]):

$$\overline{\text{kens}}(I)=\{B\,|\,B\subseteq\text{Alpha}(P_S)\;\&\;(\forall a,b\in B)\;(a,b)\notin I$$
$$\&\;(\forall c\notin B)(\exists a\in B)\;(a,c)\in I\}.$$

Assume that $\overline{\text{kens}}(I)=\{x_1,\ldots,x_n\}$.

Let us define $\mathrm{resource}(P_S) = \overline{\mathrm{kens}}(I) = \{x_1, \ldots, x_n\}$, and let $r: \mathrm{Alpha}(P_S) \to 2^{\mathrm{resource}(P_S)}$ be the function defined as follows: $(\forall a \in \mathrm{Alpha}(P_S))\ r(a) = \{x_i \mid a \in x_i\}$.

From [5] it follows that:

(1) $(\forall i = 1, \ldots, n)\ \hat{r}(x_i) = x_i$,

(2) $(\forall a, b \in \mathrm{Alpha}(P_S))\ (a, b) \in I \Leftrightarrow r(a) \cap r(b) = \emptyset$,

thus the set $\mathrm{resource}(P_S)$ and the function r are correctly defined. This construction of $\mathrm{resource}(P_S)$ and r will be applied in Section 5.2.

3. Definition of functional equivalence

3.1. Preliminaries

In order to define precisely the concept of functional equivalence we must recall some old and introduce some new notions. We start with a formal definition of *vectors of strings*.

Let $A_1, \ldots, A_n$ be alphabets, and let $A = A_1 \cup \cdots \cup A_n$. For every $i = 1, \ldots, n$, let $h_i: A^* \to A_i^*$ be a homomorphism given by

$$(\forall a \in A) \quad h_i(a) = \begin{cases} a & a \in A_i, \\ \varepsilon & a \notin A_i, \end{cases}$$

where ε denotes the empty string, and let

$$(\forall X \subseteq A^*) \quad h_i(X) = \bigcup_{x \in X} h_i(x).$$

Let us define a *concatenation* on $A_1^* \times \cdots \times A_n^*$ in the following way:

$$(\forall (x_1, \ldots, x_n), (y_1, \ldots, y_n) \in A_1^* \times \cdots \times A_n^*)$$
$$(x_1, \ldots, x_n)(y_1, \ldots, y_n) = (x_1 y_1, \ldots, x_n y_n).$$

For every $x \in A^*$, let $\mathbf{x} = (h_1(x), \ldots, h_n(x))$.

Let $\mathrm{Vect}: 2^{A^*} \to 2^{A_1^* \times \cdots \times A_n^*}$ be the following *mapping*:

$$(\forall L \subseteq A^*) \quad \mathrm{Vect}(L) = \{\mathbf{x} \mid x \in L\}.$$

Let us consider $\mathrm{Vect}(A^*) \subseteq A_1^* \times \cdots \times A_n^*$. The set $\mathrm{Vect}(A^*)$ may be called a *set of vectors of strings*. One can also prove that $\mathrm{Vect}(A^*)$ is equivalent to the set of all Mazurkiewicz traces generated by the alphabet A and the relation I defined by the covering $\{A_1, \ldots, A_n\}$ ($\mathrm{Vect}(A^*)$ is isomorphic to $A^*/\approx_I$, where $I = \mathrm{sir}(\{A_1, \ldots, A_n\})$, according to the notation of [18, 5]).

Let $\overline{\mathrm{Vect}}: 2^{A^*} \to 2^{A_1^* \times \cdots \times A_n^*}$ be the following *mapping*:

$$(\forall L \subseteq A^*) \quad \overline{\mathrm{Vect}}(L) = (h_1(L) \times \cdots \times h_n(L)) \cap \mathrm{Vect}(A^*).$$

Corollary 3.1

(1) $\mathrm{Vect}(L)=\{(x_1,\ldots,x_n)\mid(\exists x\in L)(\forall i=1,\ldots,n)\ h_i(x)=x_i\in A_i^*\}$,

(2) $\overline{\mathrm{Vect}}(L)=\{(x_1,\ldots,x_n)\mid(\exists y\in A^*)(\forall i=1,\ldots,n)\ h_i(y)=x_i\in h_i(L)\subseteq A_i^*\}$.

Corollary 3.2

$$(\forall L\subseteq A^*)\quad \mathrm{Vect}(L)\subseteq\overline{\mathrm{Vect}}(L).$$

The inclusion from Corollary 3.2 is a proper one, i.e., usually $\mathrm{Vect}(L)\neq\overline{\mathrm{Vect}}(L)$. Let us consider the following two examples. To simplify the notation we will identify regular expressions with languages generated by them.

Example 3.3. Let $A_1=\{a\}$, $A_2=\{b\}$, $A=A_1\cup A_2$, $L=(ab)^*\subseteq A^*$. Then

$$\mathrm{Vect}(L)=\{(a^k,b^k)\mid k\geqslant 0\},\qquad \overline{\mathrm{Vect}}(L)=\{(a^k,b^m)\mid k\geqslant 0,\ m\geqslant 0\},$$

so $\mathrm{Vect}(L)\subsetneq\overline{\mathrm{Vect}}(L)$.

Example 3.4. Let $A_1\{a,b\}$, $A_2=\{c,d\}$, $A=A_1\cup A_2$, $L=ab\cup cd\subseteq A^*$. Then

$$\mathrm{Vect}(L)=\{(ab,\varepsilon),(\varepsilon,cd)\},\qquad \overline{\mathrm{Vect}}(L)=\{(ab,\varepsilon),(\varepsilon,cd),(ab,cd)\},$$

so $\mathrm{Vect}(L)\subsetneq\overline{\mathrm{Vect}}(L)$.

Now we recall some basic and introduce some new concepts of the *Vector Firing Sequence Semantics for generalized paths* (see [10, 14, 20, 23]).

For every language (or regular expression, single path, generalized path) X, let $\mathrm{Alpha}(X)$ denote the alphabet of X. For every regular expression R, let $|R|$ denote the language defined by R. For every language $L\subseteq A^*$, let $\mathrm{Pref}(L)=\{x\mid(\exists y\in A^*)\ xy\in L\}$. For every set of vectors of firing sequences $V\subseteq\mathrm{Vect}(A^*)$, let $\mathrm{Pref}(V)=\{x\mid(\exists y\in A^*)\ xy\in V\}$.

Let P be a single path of the form $P=$ **path** body **end**. As was mentioned above, P can be treated as an ordinary regular expression such that $P=(\mathrm{body})^*$. It is assumed (see [14]) that the *behaviour of a single path* P is fully described by the language $\mathrm{FS}(P)$, which is called the set of firing sequences, and defined as $\mathrm{FS(P)}=\mathrm{Pref}(|P|)$. The language $|P|$ is also denoted by $\mathrm{Cyc}(P)^*$ [14], or $\mathrm{SIT}(P)^*$ [10].

Let $P=P_1\ldots P_n$ be a *generalized path. The behaviour* of $P=P_1\ldots P_n$ is described by the set of all vectors of firing sequences that might be produced by P. This set, denoted by $\mathrm{VFS}(P)$ and called the set of *vector firing sequences* of P, is defined by the following equality (see [14, 20, 10]):

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

We will show that notions FS and VFS are insufficient to describe the concept of functional equivalence (see Example 3.8). We need notions characterizing not only all system histories but also full system cycles.

Let us consider two single paths

$$P_1 = \textbf{path } 5\text{p} \;;\; \text{small} \;;\; \text{plunk } \textbf{end},$$

$$P_2 = \textbf{path } 5\text{p} \;;\; \text{small} \;;\; \text{plunk} \;;\; 5\text{p} \;;\; \text{small} \;;\; \text{plunk } \textbf{end}.$$

Of course, $\mathrm{FS}(P_1) = \mathrm{FS}(P_2)$, but P_1 and P_2 not necessarily specify equivalent systems. The first path, P_1, may be interpreted as a specification of one slot 5 penny vending machine, whilst the second path, P_2, is rather a specification of the similar machine but under the additional assumption that each customer buys two packets of biscuits. FS's and VFS's rather describe *how* a system works, but sometimes we also need a formal description of *what* a system does. To this purpose we introduce notions of results for single and generalized paths.

The *result of a single path P* is described by the language

$$\mathrm{FFS}(P) = |P|,$$

which is called the set of full firing sequences of P.

The *result of a generalized path* $P = P_1 \ldots P_n$ is described by the set of all resulting vectors of firing sequences that might be produced by P. This set, denoted by $\mathrm{VFFS}(P)$ and called the set of *vector full firing sequences* of P, is defined by the following equality:

$$\mathrm{VFFS}(P) = (\mathrm{FFS}(P_1) \times \cdots \times \mathrm{FFS}(P_n)) \cap \mathrm{Vect}(\mathrm{Alpha}(P)^*).$$

In other words, VFS describes rather a procedure, while VFFS describes an aim. Of course, knowledge about the procedure not necessarrily implies knowledge about the aim, and vice versa.

A generalized path P is said to be *adequate* (see [10, 14, 20]) iff

$$(\forall \mathbf{x} \in \mathrm{VFS}(P))(\forall a \in \mathrm{Alpha}(P))(\forall y \in \mathrm{Alpha}(P)^*)\; \mathbf{x}y\mathbf{a} \in \mathrm{VFS}(P).$$

Adequacy represents the absence of even a partial deadlock.

A generalized path P is said to be *consistent* iff $\mathrm{Pref}(\mathrm{VFFS}(P)) = \mathrm{VFS}(P)$. If P is consistent, then every history of a system leads to a proper result. The notion of consistency is very similar to the notion of periodicity introduced by Shields [22]. In fact, both concepts have the same root, but the periodicity is a stronger property. One can prove that every periodic path is consistent, but not vice versa.

An action $a \in \mathrm{Alpha}(P)$ is said to be *fireable* iff

$$(\exists x \in \mathrm{Alpha}(P)^*)\; \mathbf{xa} \in \mathrm{VFS}(P).$$

Lemma 3.5. *If P is consistent and every action from* Alpha(P) *is fireable, then P is adequate.*

Proof. Let $A = \mathrm{Alpha}(P)$. Let $\mathbf{x} \in \mathrm{VFS}(P)$. Since $\mathrm{VFS}(P) = \mathrm{Pref}(\mathrm{VFFS}(P))$, we have $(\exists y \in A^*)\; \mathbf{x}\mathbf{y} \in \mathrm{VFFS}(P)$. Let $a \in A$. Since a is fireable, $(\exists \mathbf{x}' \in \mathrm{VFS}(P))\; \mathbf{x}'\mathbf{a} \in \mathrm{VFS}(P)$.

Since P is consistent, $(\exists y' \in A^*)$ $\boldsymbol{x'ay'} \in \text{VFFS}(P)$. Note that if $\boldsymbol{x_1} \in \text{VFFS}(P)$ and $\boldsymbol{x_2} \in \text{VFFS}(P)$, then $\boldsymbol{x_1 x_2} \in \text{VFFS}(P)$. Thus, $\boldsymbol{xyx'ay'} \in \text{VFFS}(P)$, so $\boldsymbol{xyx'a} \in \text{VPS}(P)$, but this means that P is adequate. □

3.2. *The definition*

We will now return to our primary sequential single path and, derived from it, a generalized path.

Let P_S be an arbitrary, fixed for the rest of this section, single path representing sequential solution of a given problem.

Let $A = \text{Alpha}(P_S)$ be the alphabet of P_S, $R = \text{resource}(P_S)$ be a set of abstract resources associated with P_S, $r: A \to 2^R$ be the resource association function, and let $\hat{r}: R \to 2^A$ be the action distribution function. Recall that $\hat{r}$ is fully described by r, and

$$(\forall x \in R) \quad \hat{r}(x) = \{a \mid a \in A \;\&\; x \in r(a)\}.$$

Assume that $R = \text{resource}(P_S) = \{x_1, \ldots, x_n\}$. Let us put $A_i = \hat{r}(x_i)$ for $i = 1, \ldots, n$. Note that $A = A_1 \cup \cdots \cup A_n$. As was mentioned above, the behaviour of a single path P_S is described by a language $\text{FS}(P_S)$, and the result of P_S is described by a language $\text{FFS}(P_S)$. Note that $\text{FS}(P_S) = \text{Pref}(\text{FFS}(P_S))$.

To explain the intuition of the next notions we consider the following example. Let $P_S = \textbf{path } a \,;\, b \,;\, c \textbf{ end}$, $\text{resource}(P_S) = \{x_1, x_2\}$, and $r(a) = \{x_1, x_2\}$, $r(b) = \{x_1\}$, $r(c) = \{x_2\}$. Thus $A_1 = \hat{r}(x_1) = \{a, b\}$, $A_2 = \hat{r}(x_2) = \{a, c\}$. In this case we have: the behaviour of P_S, $\text{FS}(P_S) = (abc)^*(ab \cup a \cup \varepsilon) = \{\varepsilon, a, ab, abc, abca, abcab, \ldots\}$, and the result of P_S, $\text{FFS}(P_S) = (abc)^* = \{\varepsilon, abc, abcabc, \ldots\}$.

Let us reflect what kind of sequence vectors may be interpreted (in accordance with our intuition) as a *concurrent behaviour* and a *concurrent result* defined by P_S and the function r. There is no problem with the result. Note that

$$\text{Vect}(\text{FFS}(P_S)) = (\boldsymbol{abc})^* = \{\boldsymbol{\varepsilon}, \boldsymbol{abc}, \boldsymbol{abcabc}, \ldots\},$$

so the difference between $\text{Vect}(\text{FFS}(P_S))$ and $\text{FFS}(P_S)$ consists only in the fact that $\text{Vect}(\text{FFS}(P_S))$ enables one to perform independent actions concurrently.

Thus $\text{Vect}(\text{FFS}(P_S))$ may be treated as a concurrent result defined by P_S and the function r. The problem with behaviour is somewhat more complicated. The set $\text{Vect}(\text{FS}(P_S))$ looks rather strange. For instance, $\boldsymbol{abc} \in \text{Vect}(\text{FS}(P_S))$, $\boldsymbol{ab} \in \text{Vect}(\text{FS}(P_S))$, but $\boldsymbol{ac} \notin \text{Vect}(\text{FS}(P_S))$ although $\boldsymbol{abc} = \boldsymbol{acb}$!

From the notion of behaviour we usually demand that the beginning of every history is also a history (compare [21]), or, in other words, the behaviour must be closed under the operation Pref. On the other hand, the concurrent behaviour defined by P_S and r should 'approximate' $\text{Vect}(\text{FS}(P_S))$, because $\text{FS}(P_S)$ defines the behaviour of P_S and the Vect is an operation which forgets about superfluous sequentializations.

The best 'approximation' of $\text{Vect}(\text{FS}(P_S))$ closed under Pref is merely the least set containing $\text{Vect}(\text{FS}(P_S))$ and closed under Pref. One can easily prove that this set is equal to $\text{Pref}(\text{Vect}(\text{FFS}(P_S)))$. Now we come back to our general considerations.

Let us denote

$$\mathrm{VFS}(P_S, r) = \mathrm{Pref}(\mathrm{Vect}(\mathrm{FFS}(P_S))),$$

$$\mathrm{VFFS}(P_S, r) = \mathrm{Vect}(\mathrm{FFS}(P_S)).$$

We assume that the set $\mathrm{VFS}(P_S, r)$ describes the *behaviour* (concurrent) defined by the single path P_S and the resource association function r, and we assume that the set $\mathrm{VFFS}(P_S, r)$ describes the *result* (concurrent) defined by the single path P_S and the resource associated function r.

Now we may define the notion of functional equivalence.

Let P_C denote a generalized path derived from P_S and r using rules described in Section 2 of this paper, i.e., let

$$P_C = P_S/x_1 \ldots P_S/x_n.$$

A single path P_S and a generalized path P_C are said to be *functionally equivalent* if and only if:

(1) $\mathrm{VFS}(P_S, r) = \mathrm{VFS}(P_C)$,

(2) $\mathrm{VFFS}(P_S, r) = \mathrm{VFFS}(P_C)$.

In other words, P_S and P_C are functionally equivalent if they describe the same behaviour and the same result.

Note that $\mathrm{VFS}(P_C)$ and $\mathrm{VFFS}(P_C)$ can be described in terms of $\mathrm{FS}(P_S)$, $\mathrm{FFS}(P_S)$ and the mapping $\overline{\mathrm{Vect}}$.

Lemma 3.6

(1) $\mathrm{VFS}(P_C) = \overline{\mathrm{Vect}}(\mathrm{FS}(P_S))$.

(2) $\mathrm{VFFS}(P_C) = \overline{\mathrm{Vect}}(\mathrm{FFS}(P_S))$.

Proof. $\mathrm{VFS}(P_C) = (\mathrm{FS}(P_S/x_1) \times \cdots \times \mathrm{FS}(P_S/x_n)) \cap \mathrm{Vect}(A^*)$. But $\mathrm{FS}(P_S/x_i) = h_i(\mathrm{FS}(P_S))$ for $i = 1, \ldots, n$. The same holds for $\mathrm{VFFS}(P_C)$. □

Thus the functional equivalence can be formulated in terms of $\mathrm{FFS}(P_S)$, Vect and $\overline{\mathrm{Vect}}$.

Lemma 3.7. *A single path P_S and a generalized path P_C are functionally equivalent iff*:

(1) $\mathrm{Pref}(\mathrm{Vect}(\mathrm{FFs}(P_S))) = \overline{\mathrm{Vect}}(\mathrm{Pref}(\mathrm{FFS}(P_S)))$.

(2) $\mathrm{Vect}(\mathrm{FFS}(P_S)) = \overline{\mathrm{Vect}}(\mathrm{FFS}(P_S))$.

Proof. The proof follows from the fact that $\mathrm{FS}(P_S) = \mathrm{Pref}(\mathrm{FFS}(P_S))$ and by Lemma 3.6. □

It turns out that frequently the equality $\mathrm{VFS}(P_S, r) = \mathrm{VFS}(P_C)$ does not involve the equality $\mathrm{VFFS}(P_S, r) = \mathrm{VFFS}(P_C)$ and vice versa.

Let us consider the following two examples.

Example 3.8. Let

$$P_S = \textbf{path } a \ ; b \textbf{ end}, \ r(a) = \{x_1\}, \ r(b) = \{x_2\}.$$

Then

$$P_C = \textbf{system path } a \textbf{ end path } b \textbf{ end endsystem}.$$

Note that

$$\text{VFS}(P_S, r) = \text{VFS}(P_C) = (\boldsymbol{a} \cup \boldsymbol{b})^*,$$

but

$$\text{VFFS}(P_S, r) = \{(a^k, b^k) \mid k \geq 0\},$$

while

$$\text{VFFS}(P_C) = (\boldsymbol{a} \cup \boldsymbol{b})^* = \{(a^k, b^m) \mid k \geq 0, m \geq 0\},$$

so $\text{VFFS}(P_S, r) \neq \text{VFFS}(P_C)$.

Example 3.8 *shows that the notion of* VFS *is insufficient itself to describe the concept of functional equivalence. In this case,* $\text{VFS}(P_S, r) = \text{VFS}(P_C)$, *but* P_S *and* P_C *are not equivalent in the intuitive sense.*

Example 3.9. Let

$$P_S = \textbf{path } (a \ ; c \ ; e), (b \ ; d \ ; f) \textbf{ end}, \ r(a) = \{x_1\}, \ r(b) = \{x_1\}, \ r(c) = \{x_2\},$$
$$r(d) = \{x_2\}, \ r(e) = \{x_1, x_2\}, \ r(f) = \{x_1, x_2\}.$$

Then

$$\begin{aligned} P_C = \ &\textbf{system} \\ &\quad \textbf{path } (a \ ; e), (b \ ; f) \textbf{ end} \\ &\quad \textbf{path } (c \ ; e), (d \ ; f) \textbf{ end} \\ &\textbf{endsystem}. \end{aligned}$$

One may prove that

$$\text{VFFS}(P_S, r) = \text{VFFS}(P_C) = (\boldsymbol{ace} \cup \boldsymbol{bdf})^*,$$

but

$$\boldsymbol{ad} \in \text{VFS}(P_C) - \text{VFS}(P_S, r), \ \text{so} \ \text{VFS}(P_C) \neq \text{VFS}(P_S, r).$$

The property of functional equivalence implies a very regular structure of P_C.

Theorem 3.10. *If* P_C *and* P_S *are functionally equivalent, then* P_C *is consistent and every action of* P_C *is fireable.*

Proof

$$\mathrm{VFS}(P_C) = \mathrm{VFS}(P_S, r) = \mathrm{Pref}(\mathrm{Vect}(\mathrm{FFS}(P_S))) = \mathrm{Pref}(\overline{\mathrm{Vect}}(\mathrm{FFS}(P_S)))$$
$$= \mathrm{Pref}(\mathrm{VFFS}(P_S)),$$

so P_C is consistent. Let $a \in \mathrm{Alpha}(P_C) = \mathrm{Alpha}(P_S)$. Of course, a is fireable in P_S, so there is $x \in \mathrm{Alpha}(A)^*$ such that $xa \in \mathrm{FS}(P_S)$. Let $xay \in \mathrm{FFS}(P_S)$. Since $\mathrm{VFS}(P_C) = \mathrm{Pref}(\mathrm{Vect}(\mathrm{FFS}(P_S)))$, we have $xa \in \mathrm{VFS}(P_C)$, so a is also fireable in P_C. □

Corollary 3.11. *If P_C and P_S are functionally equivalent, then P_C is adequate.*

Proof. The proof follows from Theorem 3.10 and Lemma 3.5. □

The above corollary gives us a negative criterion for functional equivalence. If P_C is not adequate or if it deadlocks, then P_C and P_S are functionally different.

4. Necessary and sufficient conditions

When a sequential single path P_S is not complicated, then we can verify the functional equivalence directly from the definition, but when P_S is large, then such a procedure is a difficult and very uphill task. Unfortunately, in the general case we do only know necessary conditions, and in order to prove the functional equivalence we must use the definition.

But if we restrict our attention to paths in which the repetition of actions is restricted, then an appropriate sufficient condition can be formulated and proved.

A single path $P =$ **path** body **end** is said to be an *E^*-path* iff no action occurs more than once in body (see [14]).

Let $P = P_1 \ldots P_n$ be a generalized path.

A generalized path $P = P_1 \ldots P_n$ is said to be a *GE*-path* if every P_i $(i = 1, 2, \ldots, n)$ is an E^*-path (see [14]).

For every $a \in \mathrm{Alpha}(P)$, let $\mathrm{occ}_i(a)$ denote the number of occurrences of "a" in P_i.

For instance, if

$P =$ **system** P_1 : **path** a ; b, a **end** P_2 : **path** b, a ; b, c **end endsystem**,

then

$$\mathrm{occ}_1(\mathrm{a}) = 2, \quad \mathrm{occ}_1(b) = 1, \quad \mathrm{occ}_1(c) = 0, \quad \mathrm{occ}_2(a) = 1, \quad \mathrm{occ}_2(b) = 2, \quad \mathrm{occ}_2(c) = 1.$$

A generalized path $P = P_1 \ldots P_n$ is said to be a *GR1*-path* iff

$$(\forall a \in \mathrm{Alpha}(P))(\forall i = 1, \ldots, n) \quad \mathrm{occ}_i(a) > 1 \Rightarrow [(\forall j \neq i)\, \mathrm{occ}_j(a) \leq 1].$$

In other words, an action a may be repeated in one path only. For instance,

$P =$ **system path** a ; b ; a **end path** b, a ; b **end endsystem** is a GR1*-path, but

$$P' = \textbf{system path } a\ ;\ b\ ;\ a \textbf{ end path } a\ ;\ c\ ;\ a \textbf{ end}$$

is not a GR1*-path, because the action a occurs twice in two single paths.

Let $P_S = \textbf{path}$ body $\textbf{end}$ be a single path, and let $r: \text{Alpha}(P_S) \to r^{\text{resource}(P_S)}$ be a resource association function. Recall that the path P_S can be treated as an ordinary regular expression of the form $P_S = (\text{body})^*$.

Let us put $A = \text{Alpha}(P_S)$.

Let $I \subseteq A \times A$ be the following relation:

$$(\forall a, b \in A) \quad (a, b) \in I \iff r(a) \cap r(b) = \emptyset.$$

The relation I will be called the *independence relation. The dependence relation* is defined as $D = A \times A - I$.

Let us put $L = \text{FFS}(P_S)$.

Let $E \subseteq A \times A$ be the relation defined as follows:

$$\begin{aligned}(\forall a, b \in A) \quad (a, b) \in E \iff (\exists x \in A^*)\ & \boldsymbol{xa} \in \text{Pref}(\text{Vect}(L))\ \& \\ & \boldsymbol{xb} \in \text{Pref}(\text{Vect}(L)) \\ & \&\ \boldsymbol{xab} \notin \text{Pref}(\text{Vect}(L))\ \&\ a \neq b.\end{aligned}$$

The relation E will be called the *mutual exclusion relation.*

Every regular expression of the form $(R)^*$ or a^*, where R is a regular expression, "a" is a symbol, will be called a *starexpression.*

A symbol "a" will be called an *outer cycle generated by* a^*.

A string x is said to be an *outer cycle generated by a starexpression* $(R)^*$ iff $x \in |R'|$, where R' is derived from R by replacing all starexpressions of R by ε and removing all ε's.

Example 4.1. If $R = a \cup b(cd)^*e(g^*f)^* \cup h^*$, then after replacing all starexpresions of R by ε's we obtain $a \cup b\varepsilon e\varepsilon \cup \varepsilon$, next after removing all ε's we have $a \cup be$; so $R' = a \cup be$, and there are two outer cycles generated by $(R)^*$: a, be.

A string x is said to be a *cycle generated by a regular expression* R iff there is a starexpression $(R')^*$ included in R, i.e., $R = Q_1(R')^*Q_2$ where $Q_i \in (\text{Alpha}(R) \cup \{\cup, *, \ ), \ (\})^*$, such that x is an outer cycle generated by $(R')^*$. For instance, if R is as in Example 4.1, then R generates the following cycles: cd, g, f, h.

For every regular expression R, let C_R denote the *set of all cycles generated by R.*

For every string x, let $\text{Alpha}(x)$ denote the set of symbols occurring in x.

Let us put $\text{CD} = \{\text{Alpha}(x) \mid x \in C_{P_S}\}$. The set CD will be called the set of *cycle domains of* P_S.

Example 4.2. If $P_S = \textbf{path } a, (b\ ;\ (c\ ;\ d)^*\ ;\ e) \textbf{ end}$, then $\text{CD} = \{\{a\}, \{b, e\}, \{c, d\}\}$.

Let $P_C = P_S/x_1 \ldots P_S/x_n$ be the generalized path derived from P_S and the resource association function r.

For every relation Q, let $Q^+ = \bigcup_{i=1}^{\infty} Q^i = Q^*Q$.

Theorem 4.3 (necessary conditions for the general case). *Let P_S be a single path. If P_S and P_C are functionally equivalent, then*:

(1) $E \cap I = \emptyset$,

(2) $(\forall X \in \mathrm{CD})(\forall Y \subseteq X)$

(Y *is a maximal subset of* X *such that* $(D \cap Y \times Y)^+ = Y \times Y) \Rightarrow Y \in \mathrm{CD}$.

The second condition means that the graph of dependency relation D restricted to any cycle domain is either connected or each of its maximal connected components also creates a cycle domain.

Theorem 4.4 (sufficient conditions if P_C is a GR1*-path). *Let P_S be a single path, and let P_C be a* GR1*-*path. If*:

(1) $E \cap I = \emptyset$, *and*

(2) $(\forall X \in \mathrm{CD})\,(D \cap X \times X)^+ = X \times X$,

then P_S and P_C are functionally equivalent.

Here the second condition means that the graph of dependency relation D restricted to any cycle domain is connected.

Theorem 4.5 (necessary and sufficient conditions if P_S is an E^*-path). *Let P_S be an E^*-path. Then*: *P_S and P_C are functional equivalent if and only if*:

(1) $E \cap I = \emptyset$, *and*

(2) $(\forall X \in \mathrm{CD})\,(D \cap X \times X)^+ = X \times X$.

The proofs are long and they will be presented in a separate section (see Section 6).

5. Applications

5.1. The cigarette smokers problem

Patil [19] introduced the following synchronization problem:

"Three smokers are sitting at a table. One of them has tobacco, another has cigarette papers, and the third has matches; each one has a different ingredient required to make and smoke a cigarette but he may not give an ingredient to another. On the table in front of them, two of the three ingredients will be placed, and the smoker who has the necessary third ingredient should pick the ingredients from the table, make a cigarette and smoke it. Further ingredients are not put on the table until the old ones have been consumed. Other smokers must not interfere with the smoker who has the ingredients on the table before him. Hence co-ordination is required between the smokers."

The cigarette smokers problem was restated by Lauer and Campbell [11] in the following way:

(1) Decide which of the ingredients should be put on the table.
(2) Produce each ingredient and place it on the table.
(3) Choose the correct consumer to consumer the available ingredient.
(4) Go back to (1).

As a matter of fact, the decision which of the ingredients should be put on the table immediately indicates the correct consumer.

The final solution proposed by Lauer and Campbell [11] is the following:

```
P_LC = system
          path supplytm, supplypt ; tobacco ; m-smoker, p-smoker end
          path supplytm, supplymp ; match ; t-smoker, p-smoker end
          path supplypt, supplymp ; paper ; t-smoker, m-smoker end
       endsystem
```

where the meanings of actions are the following:
supplytm—supply tobacco and matches,
supplymp—supply matches and paper,
supplypt—supply paper and tobacco,
tobacco—tobacco on the table,
match—matches on the table,
paper—paper on the table,
m-smoker—the smoker with matches smokes,
p-smoker—the smoker with paper smokes,
t-smoker—the smoker with tobacco smokes.

A sequential solution of the cigarette smokers problem is not difficult, and it may be presented in the following form:

```
P_S = path (supplytm ; tobacco ; match ; p-smoker),
           (supplymp ; match ; paper ; t-smoker),
           (supplypt ; paper ; tobacco ; m-smoker) end
```

In this case we have three abstract resources T, P, M interpreted as

T—tobacco, P—paper, M—matches.

The resource association function is the following:

$$r(\text{supplytm}) = r(\text{p-smoker}) = \{T, M\},$$

$$r(\text{supplymp}) = r(\text{t-smoker}) = \{M, P\},$$

$$r(\text{supplypt}) = r(\text{m-smoker}) = \{P, T\},$$

$$r(\text{tobacco}) = \{T\}, \quad r(\text{Match}) = \{M\}, \quad r(\text{paper}) = \{P\}.$$

Thus $P_C = P_S/T\, P_S/M\, P_S/M$ is the following:

P_C = **system**
P_S/T : **path** (supplytm ; tobacco ; p-smoker),
(supplypt ; tobacco ; m-smoker) **end**
P_S/M : **path** (supplymp ; match ; t-smoker),
(supplytm ; match ; p-smoker) **end**
P_S/P : **path** (supplymp ; paper ; t-smoker),
(supplypt ; paper ; m-smoker) **end**
endsystem.

Note that P_C is a GR1*-path, so we can use Theorem 4.4. One can easily show that conditions (1) and (2) of Theorem 4.4 are fulfilled, so P_S and P_C are functionally equivalent.

Note that $\mathrm{VFS}(P_C) = \mathrm{VFS}(P_{LC})$, $\mathrm{VFFS}(P_C) = \mathrm{VFFS}(P_{LC})$, thus P_C and P_{LC} are equivalent in the sense of the Vector Firing Sequence Semantics. The Petri net simulating P_C (see rules in [13, 14, 11]) is simpler than the Petri net simulating P_{LC} in that sense that the first one has less conflicts.

5.2. The first reader-writer problem

The first reader-writer problem [1] may be formulated as follows (compare [12]):

"Consider a system consisting of a single resource involving read and write operations and a set of "reader" and "writer" processes which repeatedly use the operations to read from and write to the resource, respectively. It is required that any number of readers may be concurrently using the resource, but each writer must have exclusive use of it. Also, no writer may jointly use the resource with a reader. Furthermore, no reader should be kept waiting unless a writer is using the resource."

The sequential specification of that problem is trivial, and in the case of n readers and m writers it looks as follows:

P_S = **path** $\mathrm{read}_1, \mathrm{read}_2, \ldots, \mathrm{read}_n, \mathrm{write}_1, \mathrm{write}_2, \ldots, \mathrm{write}_m$ **end,**

where the interpretation of actions is fully described by their names.

In the case of a noisy vending machine and cigarette smokers, the set of abstract resources corresponded to real system resources. In this case, we have only one real resource, so the set of abstract resources must be defined in a different way. We recall that an abstract resource may be associated with a set of actions which, for various reasons, must be performed only one at a time.

Note that in this case the independence relation I, i.e., the relation describing which actions may be performed concurrently, can easily be described on the basis of the problem formulation.

Namely:

$$I = \{\mathrm{read}_i, \mathrm{read}_j) \mid i \neq j\}.$$

The family $\overline{\text{kens}}(I)$ defined by the relation I is of the form

$$\overline{\text{kens}}(I) = \{\{\text{write}_1, \ldots, \text{write}_m, \text{read}_1\}, \ldots, \{\text{write}_1, \ldots, \text{write}_m, \text{read}_n\}\}.$$

Let us put

$$x_i = \{\text{write}_1, \ldots, \text{write}_m, \text{read}_i\} \quad \text{for } i = 1, \ldots, n.$$

Thus resource $(P_S) = \overline{\text{kens}}(I) = \{x_1, \ldots, x_n\}$, and

$$(\forall i = 1, \ldots, n) \quad r(\text{read}_i) = \{x_i\},$$
$$(\forall j = 1, \ldots, m) \quad r(\text{write}_j) = \{x_1, \ldots, x_n\}, \quad \text{and}$$
$$(\forall i = 1, \ldots, n) \quad \hat{r}(x_i) = x_i.$$

Next, using the standard procedure from Section 2 we may obtain $P_C = P_S/x_1 \ldots P_S/x_n$, which is of the following form:

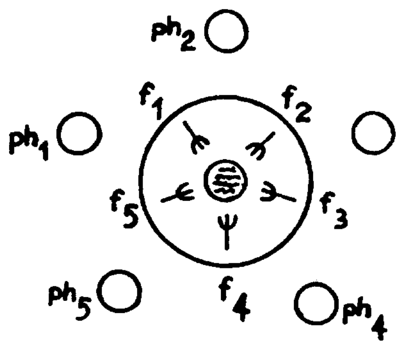
$P_C =$ **system**
  P_S/x_1 : **path** write$_1, \ldots,$ write$_m$, read$_1$ **end**
  ....
  P_S/x_n : **path** write$_1, \ldots,$ write$_m$, read$_n$ **end**
  endsystem.

In this case, P_S is an E^*-path, so we can use Theorem 4.5. One can easily verify that P_S and P_C are functionally equivalent. Note that P_C is identical with a solution presented in [12].

5.3. Dining philosophers

Now we consider the standard synchronization problem consisting of five philosophers who alternately think or eat [2]. To eat, a philosopher needs two forks, but unfortunately there are only five forks on the circular table and each philosopher is only allowed to use the two forks nearest to him. Obviously, two neighbours cannot eat at the same time. Essentially, this is a resource allocation problem.

Assume that the philosophers and forks are numbered in the following way:



This system may involve the following actions:

e_i—the ith philosopher eats,
pufl_i—the ith philosopher picks up a fork by his left hand,
pufr_i—the ith philosopher picks up a fork by his right hand,
pdfl_i—the ith philosopher puts down his left fork,
pdfr_i—the ith philosopher puts down his right fork,

where $i = 1, \ldots, 5$.

The sequential solution is also very easy, and it can be presented by the following single path:

$$\begin{aligned} P_S = \textbf{path}\ & (\mathrm{pufl}_1 ; \mathrm{pufr}_1 ; e_1 ; \mathrm{pdfl}_1 ; \mathrm{pdfr}_1), \\ & (\mathrm{pufl}_2 ; \mathrm{pufr}_2 ; e_2 ; \mathrm{pdfl}_2 ; \mathrm{pdfr}_2), \\ & (\mathrm{pufl}_3 ; \mathrm{pufr}_3 ; e_3 ; \mathrm{pdfl}_3 ; \mathrm{pdfr}_3), \\ & (\mathrm{pufl}_4 ; \mathrm{pufr}_4 ; e_4 ; \mathrm{pdfl}_4 ; \mathrm{pdfr}_4), \\ & (\mathrm{pufl}_5 ; \mathrm{pufr}_5 ; e_5 ; \mathrm{pdfl}_5 ; \mathrm{pdfr}_5)\ \textbf{end}. \end{aligned}$$

In this case we can distinguish five abstract resources:

f_i—the ith fork, $i = 1, 2, 3, 4, 5$.

The resource association function r is of the following form. For every $i = 1, 2, 3, 4, 5$:

$$r(e_i) = \{f_i, f_{i\ominus 1}\},$$

$$r(\mathrm{pufl}_i) = r(\mathrm{pdfl}_i) = \{f_i\},$$

$$r(\mathrm{pufr}_i) = r(\mathrm{pdfr}_i) = \{f_{i\ominus 1}\},$$

where $i \ominus 1 := \textbf{if}\ i > 1\ \textbf{then}\ i - 1\ \textbf{else}\ 5$.

The single path P_S and the function r define the following generalized path P_C:

$$\begin{aligned} P_C = \ & \textbf{system} \\ & P_S/f_1 : \textbf{path}\ (\mathrm{pufl}_1 ; e_1 ; \mathrm{pdfl}_1), (\mathrm{pufr}_2 ; e_2 ; \mathrm{pdfr}_2)\ \textbf{end} \\ & P_S/f_2 : \textbf{path}\ (\mathrm{pufl}_2 ; e_2 ; \mathrm{pdfl}_2), (\mathrm{pufr}_3 ; e_3 ; \mathrm{pdfr}_3)\ \textbf{end} \\ & P_S/f_3 : \textbf{path}\ (\mathrm{pufl}_3 ; e_3 ; \mathrm{pdfl}_3), (\mathrm{pufr}_4 ; e_4 ; \mathrm{pdfr}_4)\ \textbf{end} \\ & P_S/f_4 : \textbf{path}\ (\mathrm{pufl}_4 ; e_4 ; \mathrm{pdfl}_4), (\mathrm{pufr}_5 ; e_5 ; \mathrm{pdfr}_5)\ \textbf{end} \\ & P_S/f_5 : \textbf{path}\ (\mathrm{pufl}_5 ; e_5 ; \mathrm{pdfl}_5), (\mathrm{pufr}_1 ; e_1 ; \mathrm{pdfr}_1)\ \textbf{end} \\ & \textbf{endsystem}. \end{aligned}$$

Unfortunately, the paths P_S and P_C are *functionally different*. One can use Theorem 4.3 and show that for instance $(\mathrm{pufl}_1, \mathrm{pufl}_2) \in E \cap I$. One can also prove that, for instance, $\mathbf{pufl}_1\, \mathbf{pufl}_2\, \mathbf{pufl}_3\, \mathbf{pufl}_4\, \mathbf{pufl}_5 \in \mathrm{VFS}(P_C) - \mathrm{VFS}(P_S, r)$. Moreover, P_C deadlocks after the performance of the sequence $\mathbf{pufl}_1 \ldots \mathbf{pufl}_5$, while P_S is obviously adequate.

Let us observe that while decomposing P_S into P_C we lose the information that when the ith philosopher is going to eat, the philosophers $i \oplus 1$ and $i \ominus 1$ must think ($i \oplus 1 := \textbf{if}\ i < 5\ \textbf{then}\ i + 1\ \textbf{else}\ 1$).

For every $i = 1, 2, 3, 4, 5$, let lrf_i denote the action interpreted as the beginning of a state "both, left, and right, forks of the ith philosophers are on the table", and let $r(\mathrm{lrf}_i) = \{f_i, f_{i\ominus 1}\}$.

The new sequential solution is the following:

$$P'_S = \textbf{path}\ (\mathrm{lrf}_1\ ;\ \mathrm{pufl}_1\ ;\ \mathrm{pufr}_1\ ;\ e_1\ ;\ \mathrm{pdfl}_1\ ;\ \mathrm{pdfr}_1),$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$(\mathrm{lrf}_5\ ;\ \mathrm{pufl}_5\ ;\ \mathrm{pufr}_5\ ;\ e_5\ ;\ \mathrm{pdfl}_5\ ;\ \mathrm{pdfr}_5)\ \textbf{end}.$$

Of course, from the sequential viewpoint, P_S and P'_S are essentially the same. The single path P'_S and the function r define the following generalized path P'_C:

$P'_C = \textbf{system}$

$P_S/f_1 : \textbf{path}\ (\mathrm{lrf}_1\ ;\ \mathrm{pufl}_1\ ;\ e_1\ ;\ \mathrm{pdfl}_1), (\mathrm{lrf}_2\ ;\ \mathrm{pufr}_2\ ;\ e_2\ ;\ \mathrm{pdfr}_2)\ \textbf{end}$

$P_S/f_2 : \textbf{path}\ (\mathrm{lrf}_2\ ;\ \mathrm{pufl}_2\ ;\ e_2\ ;\ \mathrm{pdfl}_2), (\mathrm{lrf}_3\ ;\ \mathrm{pufr}_3\ ;\ e_3\ ;\ \mathrm{pdfr}_3)\ \textbf{end}$

$P_S/f_3 : \textbf{path}\ (\mathrm{lrf}_3\ ;\ \mathrm{pufl}_3\ ;\ e_3\ ;\ \mathrm{pdfl}_3), (\mathrm{lrf}_4\ ;\ \mathrm{pufr}_4\ ;\ e_4\ ;\ \mathrm{pdfr}_4)\ \textbf{end}$

$P_S/f_4 : \textbf{path}\ (\mathrm{lrf}_4\ ;\ \mathrm{pufl}_4\ ;\ e_4\ ;\ \mathrm{pdfl}_4), (\mathrm{lrf}_5\ ;\ \mathrm{pufr}_5\ ;\ e_5\ ;\ \mathrm{pdfr}_5)\ \textbf{end}$

$P_S/f_5 : \textbf{path}\ (\mathrm{lrf}_5\ ;\ \mathrm{pufl}_5\ ;\ e_5\ ;\ \mathrm{pdfl}_5), (\mathrm{lrf}_1\ ;\ \mathrm{pufr}_1\ ;\ e_1\ ;\ \mathrm{pdfr}_1)\ \textbf{end}$

endsystem.

Note that P'_S is an E^*-path, so we can use Theorem 4.5. One can prove (although in this case it is a somewhat uphill task) that now $E \cap I = \emptyset$ and $(\forall X \in \mathrm{CD})$ $(D \cap X \times X)^+ = X \times X$, so P'_S and P'_C are functionally equivalent.

In this case, the introduction of the actions 'lrf_i' does patch up the solution. This introduction was not suggested by the sequential solution, to which, as was stated above, they make no substantial difference; this introduction was suggested by analysing reasons for which P_C and P_S turned out to be functionally different.

6. The proof of necessary and sufficient conditions

At first we prove Theorems 4.3 and 4.5, and then Theorem 4.4. Proofs of Theorems 4.3 and 4.5 are by induction on the form of regular expressions, and they consist of a number of auxiliary lemmas.

Let $A, A_1, \ldots, A_n$ be alphabets, and let $A = A_1 \cup \cdots \cup A_n$. By a regular expression we will understand a regular expressions under A.

A regular expression R is said to be an E^*-expression if there is no symbol occurring more than once in R.

In this section, for every $x \in A^*$, every $L \subseteq A^*$ and every regular expression R under A, the symbols $A(x)$, $A(L)$ or $A(R)$ will denote the set of all symbols occurring in x, L or R. Writing Vect and $\overline{\mathrm{Vect}}$ we will understand that $\mathrm{Vect}, \overline{\mathrm{Vect}} : 2^{A^*} \to 2^{A_1^* \times \cdots \times A_n^*}$.

For every $V_1, V_2 \subseteq \mathrm{Vect}(A^*)$, let

$$V_1 V_2 = \{\boldsymbol{xy} \mid \boldsymbol{x} \in V_1\ \&\ \boldsymbol{y} \in V_2\},$$

$$(V_1)^* = \{\boldsymbol{x}_1 \ldots \boldsymbol{x}_k \mid \boldsymbol{x}_i \in V_1\ \&\ k \geq 0\}.$$

From the definition of $\overline{\text{Vect}}$ we obtain the following results.

Corollary 6.1

(1) $L_1 \subseteq L_2 \Rightarrow \overline{\text{Vect}}(L_1) \subseteq \overline{\text{Vect}}(L_2)$.
(2) $\overline{\text{Vect}}(L_1)\overline{\text{Vect}}(L_2) \subseteq \overline{\text{Vect}}(L_1L_2)$.
(3) $\overline{\text{Vect}}(L_1) \cup \overline{\text{Vect}}(L_2) \subseteq \overline{\text{Vect}}(L_1 \cup L_2)$.
(4) $\text{Pref}(\overline{\text{Vect}}(L)) \subseteq \overline{\text{Vect}}(\text{Pref}(L))$.

For every $B \subseteq A$, let $h_B: A^* \to B^*$ be the homomorphism given by

$$(\forall a \in A) \quad h_B(a) = \begin{cases} a & a \in B, \\ \varepsilon & a \notin B. \end{cases}$$

Let $I \subseteq A \times A$ be the following relation:

$$(\forall a, b \in A) \quad (a, b) \in I \Leftrightarrow (\forall i = 1, \ldots, n)\; a \notin A_i \text{ or } b \notin A_i.$$

If resource$(P_S) = \{x_1, \ldots, x_n\}$ and $A_i = \hat{r}(x_i)$, then

$$(a, b) \in I \Leftrightarrow r(a) \cap r(b) = \emptyset.$$

Let $D \subseteq A \times A$ be the relation defined as $D = A \times A - I$. For every regular expression R, let $E_R \subseteq A \times A$ be the following relation (see the definition of E in Section 4):

$$(a, b) \in E_R \Leftrightarrow (\exists x \in A^*)\; \boldsymbol{xa} \in \text{Pref}(\text{Vect}(|R|))$$
$$\& \; \boldsymbol{xb} \in \text{Pref}(\text{Vect}(|R|)) \;\&\; \boldsymbol{xab} \notin \text{Pref}(\text{Vect}(|R|)) \;\&\; a \neq b.$$

Recall that, for every regular expression R, the symbol C_R denotes the set of all cycles generated by R (see Section 4). For every regular expression R, let

$$\text{CD}_R = \{A(x) \mid x \in C_R\}.$$

At present we can formulate the basic result of this paper.

Theorem 6.2. *Let R be an E^*-expression, and let $L = |R|$. Then*

$$(\text{Vect}(L) = \overline{\text{Vect}}(L) \;\&\; \text{Pref}(\text{Vect}(L)) = \overline{\text{Vect}}(\text{Pref}(L))) \Leftrightarrow$$
$$\Leftrightarrow (E_R \cap I = \emptyset \;\&\; (\forall X \in \text{CD}_R)\,(D \cap X \times X)^+ = X \times X).$$

Theorem 4.5 is a direct consequence of Theorem 6.2.

Theorem 6.2 is somewhat more general than Theorem 4.5 because here we do not assume that R is of the form $(R')^*$.

The proof of Theorem 6.2 is by induction on the structure of a regular expression.

For $R = \varepsilon$, and $R = a$, where "a" is a symbol, the theorem is obviously true. Now we will prove that it is also true for $R = R_1R_2$.

Lemma 6.3. *Let $L_1, L_2 \subseteq A^*$ and let $A(L_1) \cap A(L_2) = \emptyset$. Then we have* $\overline{\text{Vect}}(L_1L_2) = \overline{\text{Vect}}(L_1)\,\overline{\text{Vect}}(L_2)$.

Proof

$$\begin{aligned}\overline{\mathrm{Vect}}(L_1L_2) &= \{(x_1,\dots,x_n) \mid (\exists x\in A^*)\ h_i(x)=x_i\in h_i(L_1L_2)\}\\ &= \{(y_1z_1,\dots,y_nz_n) \mid (\exists x\in A^*)\ h_i(x)=y_iz_i\\ &\qquad \&\ y_i\in h_i(L_1)\ \&\ z_i\in h_i(L_2)\}\\ &= \{(y_1,\dots,y_n)(z_1,\dots,z_n) \mid (\exists x\in A^*)\ h_i(x)=y_iz_i\\ &\qquad \&\ y_i\in h(L_1)\ \&\ z_i\in h_i(L_2)\} = V.\end{aligned}$$

Let x' denote a projection of x on $A(L_1)$, and let x'' denote a projection of x on $A(L_2)$.

Because $A(L_1)\cap A(L_2)=\emptyset$, we have $h_i(x')=y_i$, $h_i(x'')=z_i$ for $i=1,\dots,n$. Thus we can write

$$\begin{aligned}V=\{(y_1,\dots,y_n)(z_1,\dots,z_n) \mid (\exists x',x''\in A^*)\ h_i(x')=y_i\in h_i(L_1)\\ \&\ h_i(x'')=z_i\in h_i(L_2)\}=\overline{\mathrm{Vect}}(L_1)\,\overline{\mathrm{Vect}}(L_2). \qquad \square\end{aligned}$$

Lemma 6.4. *Let* $L_1, L_2\subseteq A^*$, $A(L_1)\cap A(L_2)=\emptyset$, *and let* $\mathrm{Vect}(L_i)=\overline{\mathrm{Vect}}(L_i)$ *for* $i=1,2$. *Then*: $\mathrm{Vect}(L_1L_2)=\overline{\mathrm{Vect}}(L_1L_2)$.

Proof. By the definition of Vect we have: $\mathrm{Vect}(L_1L_2)=\mathrm{Vect}(L_1)\,\mathrm{Vect}(L_2)$, and by Lemma 6.3: $\overline{\mathrm{Vect}}(L_1)\,\overline{\mathrm{Vect}}(L_2)=\overline{\mathrm{Vect}}(L_1L_2)$.

Thus we can write

$$\mathrm{Vect}(L_1L_2)=\mathrm{Vect}(L_1)\,\mathrm{Vect}(L_2)=\overline{\mathrm{Vect}}(L_1)\,\overline{\mathrm{Vect}}(L_2)=\overline{\mathrm{Vect}}(L_1L_2). \qquad \square$$

Lemma 6.5. *Let* $L_1, L_2\subseteq A^*$, *let* $A(L_1)\cap A(L_2)=\emptyset$, *and let* $\overline{\mathrm{Vect}}(\mathrm{Pref}(L_i))=\mathrm{Pref}(\overline{\mathrm{Vect}}(L_i))$ *for* $i=1,2$. *Then*: $\overline{\mathrm{Vect}}(\mathrm{Pref}(L_1L_2))=\mathrm{Pref}(\overline{\mathrm{Vect}}(L_1L_2))$.

Proof. By Corollary 6.1(4) we have $\mathrm{Pref}(\overline{\mathrm{Vect}}(L_1L_2))\subseteq\overline{\mathrm{Vect}}(\mathrm{Pref}(L_1L_2))$.

Let $x=(x_1,\dots,x_n)\in\overline{\mathrm{Vect}}(\mathrm{Pref}(L_1L_2))$. This means that

$$(\exists y\in A^*)(\forall i=1,\dots,n)\quad h_i(y)=x_i\in h_i(\mathrm{Pref}(L_1L_2)).$$

Note that

$$h_i(\mathrm{Pref}(L_1L_2))=h_i(\mathrm{Pref}(L_1))\cup h_i(L_1)h_i(\mathrm{Pref}^{\cdot}(L_2)),$$

where $\mathrm{Pref}^{\cdot}(L_2)=\mathrm{Pref}(L_2)-\{\varepsilon\}$.

Thus every x_i can be represented in the form: $x_i=y_iz_i$, where

$$(y_i\in h_i(\mathrm{Pref}(L_1))\ \&\ z_i=\varepsilon)\ \text{or}\ (y_i\in h_i(L_1)\ \&\ z_i\in h_i(\mathrm{Pref}^{\cdot}(L_2))).$$

Let us consider $(y_1,\dots,y_n)$. Since $A(L_1)\cap A(L_2)=\emptyset$, we have

$$y_i=h_{A(L_1)}(y_iz_i)=h_{A(L_1)}(x_i)=h_{A(L_1)}(h_i(x))=h_i(h_{A(L_1)}(x)),$$

for all i. Thus, we have $y_i=h(y')$, for all i, where $y'=h_{A(L_1)}(x)$. Thus $(y_1,\dots,y_n)\in\overline{\mathrm{Vect}}(\mathrm{Pref}(L_1))$.

Similarly we can show that $(z_1, \ldots, z_n) \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L_2))$. Recall that $y' = h_{A(L_1)}(x)$; et $z' = h_{A(L_2)}(x)$.

Since $\overline{\mathrm{Vect}}(\mathrm{Pref}(L_i)) = \mathrm{Pref}(\overline{\mathrm{Vect}}(L_i))$, $i = 1, 2$, we have

$$y' = (y_1, \ldots, y_n) \in \mathrm{Pref}(\overline{\mathrm{Vect}}(L_1)) \quad \text{and} \quad z' = (z_1, \ldots, z_n) \in \mathrm{Pref}(\overline{\mathrm{Vect}}(L_2)).$$

.et y'' be a shortest string such that $y'y'' \in \overline{\mathrm{Vect}}(L_1)$, and let z'' be a shortest string uch that $z'z'' \in \overline{\mathrm{Vect}}(L_2)$. For all $i = 1, \ldots, n$, let $y'_i = h_i(y'')$, $z'_i = h_i(z'')$. Note that $'_i = \varepsilon$ or $z_i = \varepsilon$ for every $i = 1, \ldots, n$. Let us put $x' = y'y''z'z''$.

Of course, $(\forall i = 1, \ldots, n)\ h_i(x') = y_i y'_i z_i z'_i \in h_i(L_1 L_2)$, so $x' \in \overline{\mathrm{Vect}}(L_1 L_2)$. But since $'_i = \varepsilon$ or $z_i = \varepsilon$, we have $h_i(x') = y_i y'_i z_i z'_i = y_i z_i y'_i z'_i$. But this means that $x \in \mathrm{Pref}(\{x'\}) \subseteq$ 'ref$(\overline{\mathrm{Vect}}(L_1 L_2))$. □

.emma **6.6.** *Let* $L_1, L_2 \subseteq A^*$, $A(L_1) \cap A(L_2) = \emptyset$, $\mathrm{Vect}(L_i) = \overline{\mathrm{Vect}}(L_i)$ *and let* 'ref$(\mathrm{Vect}(L_i)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L_i))$ *for* $i = 1, 2$. *Then*:

$$\mathrm{Vect}(L_1 L_2) = \overline{\mathrm{Vect}}(L_1 L_2) \text{ and } \mathrm{Pref}(\mathrm{Vect}(L_1 L_2)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L_1 L_2)).$$

'roof. The equality $\mathrm{Vect}(L_1 L_2) = \overline{\mathrm{Vect}}(L_1 L_2)$ follows from Lemma 6.4. From Lemma .5 we have $\overline{\mathrm{Vect}}(\mathrm{Pref}(L_1 L_2)) = \mathrm{Pref}(\overline{\mathrm{Vect}}(L_1 L_2))$. But from those two statements it ollows that $\mathrm{Pref}(\mathrm{Vect}(L_1 L_2)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L_1 L_2))$. □

The above lemma proves that Theorem 6.2 is true for $R = R_1 R_2$. The next case /e must prove is the case that $R = R_1 \cup R_2$.

emma 6.7. *Let* R_1, R_2, $R = R_1 \cup R_2$ *be* E^**-expressions. Let* $L_i = |R_i|$, $\mathrm{Pref}(\mathrm{Vect}(L_i)) =$ $\overline{\text{ect}}(\mathrm{Pref}(L_i))$ *for* $i = 1, 2$. *Let* $E_R \cap I = \emptyset$.

Then: $\mathrm{Pref}(\mathrm{Vect}(L_1 \cup L_2)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L_1 \cup L_2))$.

'roof. Let us put $L = L_1 \cup L_2$. By Corollaries 6.1 and 3.2 we have $\mathrm{Pref}(\mathrm{Vect}(L)) \subseteq$ $\overline{\text{ect}}(\mathrm{Pref}(L))$.

The opposite inclusion will be proved by contradiction. We assume that $\overline{\text{ect}}(\mathrm{Pref}(L)) - \mathrm{Pref}(\mathrm{Vect}(L)) \neq \emptyset$, and then we show that $E_R \cap I \neq \emptyset$.

Let $\boldsymbol{xab}$ be the following vector of sequences:

(1) $\boldsymbol{xab} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L)) - \mathrm{Pref}(\mathrm{Vect}(L))$,

(2) $(\forall \boldsymbol{y} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L)))\ \mathrm{length}(\boldsymbol{y}) < \mathrm{length}(\boldsymbol{xab}) \Rightarrow \boldsymbol{y} \in \mathrm{Pref}(\mathrm{Vect}(L))$.

lote that if $\overline{\mathrm{Vect}}(\mathrm{Pref}(L)) \neq \mathrm{Pref}(\mathrm{Vect}(L))$, then there exist minimal vectors of $\overline{\text{ect}}(\mathrm{Pref}(L)) - \mathrm{Pref}(\mathrm{Vect}(L))$. They must differ from $\boldsymbol{b}$, $b \in A(L)$, because in that ase we would have $b \in A(L_1) \cap A(L_2)$, contradicting the hypothesis that R is an *-expression. Thus, there always exist such a vector of sequences $\boldsymbol{xab}$. We have $\boldsymbol{ab} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L)) - \mathrm{Pref}(\mathrm{Vect}(L))$, and $\boldsymbol{xa} \in \mathrm{Pref}(\mathrm{Vect}(L))$. Because R_1, R_2, $R =$ $_1 \cup R_2$ are E^*-expressions, $A(L_1) \cap A(L_2) = \emptyset$. Since $A(L_1) \cap A(L_2) = \emptyset$, we have

$$\begin{aligned}\mathrm{Pref}(\mathrm{Vect}(L)) &= \mathrm{Pref}(\mathrm{Vect}(L_1 \cup L_2)) = \mathrm{Pref}(\mathrm{Vect}(L_1) \cup \mathrm{Vect}(L_2)) \\ &= \mathrm{Pref}(\mathrm{Vect}(L_1)) \cup \mathrm{Pref}(\mathrm{Vect}(L_2)).\end{aligned}$$

But $\text{Pref}(\text{Vect}(L_i)) = \overline{\text{Vect}}(\text{Pref}(L_i))$ for $i = 1, 2$, so

$$\text{Pref}(\text{Vect}(L)) = \overline{\text{Vect}}(\text{Pref}(L_1)) \cup \overline{\text{Vect}}(\text{Pref}(L_2)).$$

Thus, $\boldsymbol{xa} \in \overline{\text{Vect}}(\text{Pref}(L_1)) \cup \overline{\text{Vect}}(\text{Pref}(L_2))$. Because $A(L_1) \cap A(L_2) = \emptyset$, we have $\overline{\text{Vect}}(\text{Pref}(L_1)) \cap \overline{\text{Vect}}(\text{Pref}(L_2)) = \{\boldsymbol{\varepsilon}\}$.

Assume that $\boldsymbol{xa} \in \overline{\text{Vect}}(\text{Pref}(L_1))$. This means that $(\forall i = 1, \ldots, n)$ $h_i(xa) \in h_i(\text{Pref}(L_1))$. Since $\boldsymbol{xab} \in \overline{\text{Vect}}(\text{Pref}(L_1 \cup L_2))$ and $A(L_1) \cap A(L_2) = \emptyset$, we have

$$(\forall i = 1, \ldots, n) \quad h_i(xab) \in h_i(\text{Pref}(L_1 \cup L_2)) = h_i(\text{Pref}(L_1)) \cup h_i(\text{Pref}(L_2)).$$

Note that $\boldsymbol{xa} \in \overline{\text{Vect}}(\text{Pref}(L_1))$ & $\boldsymbol{xab} \in \overline{\text{Vect}}(\text{Pref}(L_1)) \cup \overline{\text{Vect}}(\text{Pref}(L_2))$ implies that $(\exists i)$ $h_i(xab) \in h_i(\text{Pref}(L_2))$.

Assume that $(\exists j)$ $\{a, b\} \subseteq A_j$. This means that $h_j(xab) = x'ab$. Since b occurs in $x'ab$, we have $x'ab \notin h_j(\text{Pref}(L_1))$, so $x'ab \in h_j(\text{Pref}(L_2))$. Thus $a \in A(L_2)$. On the other hand, $\boldsymbol{xa} \in \overline{\text{Vect}}(\text{Pref}(L_1))$, so $a \in A(L_1)$. But $A(L_1) \cap A(L_2) = \emptyset$, and the assumption $(\exists j)$ $\{a, b\} \subseteq A_j$ 'leads to a discrepancy. Thus $(a, b) \in I$, and of course $\boldsymbol{xab} = \boldsymbol{xba}$.

Since $\overline{\text{Vect}}(\text{Pref}(L)) = \text{Pref}(\overline{\text{Vect}}(\text{Pref}(L)))$, we have $\boldsymbol{xb} \in \overline{\text{Vect}}(\text{Pref}(L))$. But $\text{length}(xb) < \text{length}(xab)$, so $\boldsymbol{xb} \in \text{Pref}(\text{Vect}(L))$.

In this way we have proved that

$$\boldsymbol{xa} \in \text{Pref}(\text{Vect}(L)) \;\&\; \boldsymbol{xb} \in \text{Pref}(\text{Vect}(L)) \;\&\; \boldsymbol{xab} \notin \text{Pref}(\text{Vect}(L)),$$

so $(a, b) \in E_R$.

We have also proved that $(a, b) \in I$, thus $E_R \cap I \neq \emptyset$—in spite of the assumption. □

Lemma 6.8. *Let R_1, R_2, $R = R_1 \cup R_2$ be E^*-expressions. Let $L_i = |R_i|$,* $\text{Vect}(L_1) = \overline{\text{Vect}}(L_i)$, $\text{Pref}(\text{Vect}(L_i)) = \overline{\text{Vect}}(\text{Pref}(L_i))$ *for $i = 1, 2$. Let $E_R \cap I = \emptyset$.*

Then, $\text{Vect}(L_1 \cup L_2) = \overline{\text{Vect}}(L_1 \cup L_2)$.

Proof. By Corollary 6.1(3) we have $\overline{\text{Vect}}(L_1) \cup \overline{\text{Vect}}(L_2) \subseteq \overline{\text{Vect}}(L_1 \cup L_2)$.

From Lemma 6.7 and the proof of Lemma 6.7 we obtain

$$\overline{\text{Vect}}(\text{Pref}(L_1 \cup L_2)) = \text{Pref}(\text{Vect}(L_1 \cup L_2)) = \overline{\text{Vect}}(\text{Pref}(L_1)) \cup \overline{\text{Vect}}(\text{Pref}(L_2)).$$

$$\boldsymbol{x} \in \overline{\text{Vect}}(L_1 \cup L_2) \Rightarrow \boldsymbol{x} \in \overline{\text{Vect}}(\text{Pref}(L_1 \cup L_2)) \Rightarrow \boldsymbol{x} \in \overline{\text{Vect}}(\text{Pref}(L_1)) \cup \overline{\text{Vect}}(\text{Pref}(L_2)).$$

Assume that $\boldsymbol{x} \in \overline{\text{Vect}}(\text{Pref}(L_1))$. Since $A(L_1) \cap A(L_2) = \emptyset$, this means that $\boldsymbol{x} \in A(L_1)^*$. Thus $\boldsymbol{x} \in \overline{\text{Vect}}(L_1 \cup L_2) \cap \overline{\text{Vect}}(\text{Pref}(L_1))$. But because $A(L_1) \cap A(L_2) = \emptyset$, we have $\overline{\text{Vect}}(L_1 \cup L_2) \cap \overline{\text{Vect}}(\text{Pref}(L_1)) = \overline{\text{Vect}}(L_1)$. similarly for $\boldsymbol{x} \in \overline{\text{Vect}}(\text{Pref}(L_2))$.

In this way we have proved that $\boldsymbol{x} \in \overline{\text{Vect}}(L_1) \cup \overline{\text{Vect}}(L_2)$. Thus,

$$\overline{\text{Vect}}(L_1 \cup L_2) = \overline{\text{Vect}}(L_1) \cup \overline{\text{Vect}}(L_2) = \text{Vect}(L_1) \cup \text{Vect}(L_2) = \text{Vect}(L_1 \cup L_2). \qquad \square$$

At this point we proved the implication $\Leftarrow$ for $R = R_1 \cup R_2$.

Lemma 6.9. *Let* $\boldsymbol{xa} \in \mathrm{Pref}(\mathrm{Vect}(L))$ & $\boldsymbol{xb} \in \mathrm{Pref}(\mathrm{Vect}(L))$ & $(a, b) \in I$.
Then, $\boldsymbol{xab} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L))$.

Proof. By Corollaries 3.2 and 6.1 we have $\mathrm{Pref}(\mathrm{Vect}(L)) \subseteq \overline{\mathrm{Vect}}(\mathrm{Pref}(L))$.

$$\begin{aligned}&\boldsymbol{xa} \in \mathrm{Pref}(\mathrm{Vect}(L)) \;\&\; \boldsymbol{xb} \in \mathrm{Pref}(\mathrm{Vect}(L))\\ &\quad\Rightarrow\; (\forall i = 1, \ldots, n)\; h_i(xa) \in h_i(\mathrm{Pref}(L)) \;\&\; h_i(xb) \in h_i(\mathrm{Pref}(L)).\end{aligned}$$

Because $(a, b) \in I$, if $a \in A_i$, then $b \notin A_i$ and vice versa.

Thus, $(\forall i = 1, \ldots, n)$ if $a \in A_i$, then $h_i(xab) = h_i(xa)$, and if $b \in A_i$, then $h_i(xab) = h_i(xb)$. Let us consider $\boldsymbol{xab}$. From the above considerations it follows that

$$(\forall i = 1, \ldots, n) \quad h_i(xab) \in h_i(\mathrm{Pref}(L)), \text{ so } \boldsymbol{xab} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L)). \qquad \square$$

Lemma 6.10. *Let R be any regular expression and let* $L = |R|$.
Then, $\mathrm{Pref}(\mathrm{Vect}(L)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L)) \Rightarrow E_R \cap I = \emptyset$.

Proof. Assume that $(a, b) \in E_R \cap I$. By the definition of E_R we have

$$(\exists x) \quad \boldsymbol{xa} \in \mathrm{Pref}(\mathrm{Vect}(L)) \;\&\; \boldsymbol{xb} \in \mathrm{Pref}(\mathrm{Vect}(L)) \;\&\; \boldsymbol{xab} \notin \mathrm{Pref}(\mathrm{Vect}(L)).$$

By Lemma 6.9 we have: $\boldsymbol{xab} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L))$.

Thus, $\mathrm{Pref}(\mathrm{Vect}(L)) \neq \overline{\mathrm{Vect}}(\mathrm{Pref}(L))$. $\square$

The above lemma proves the implication $\Rightarrow$ for $R = R_1 \cup R_2$, thus Theorem 6.2 is true for $R = R_1 \cup R_2$. To prove the whole theorem we must show its truthfulness for $R = (R_1)^*$.

Lemma 6.11

$$\mathrm{Pref}(\mathrm{Vect}(L)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L)) \;\Rightarrow\; \mathrm{Pref}(\mathrm{Vect}(L^*)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L^*)).$$

Proof. The proof follows by induction on the length of $\boldsymbol{x}$ from $\overline{\mathrm{Vect}}(\mathrm{Pref}(L^*))$.

From Corollaries 3.2 and 6.1, we have $\mathrm{Pref}(\mathrm{Vect}(L^*)) \subseteq \overline{\mathrm{Vect}}(\mathrm{Pref}(L^*))$. Note that $\boldsymbol{\varepsilon} \in \mathrm{Pref}(\mathrm{Vect}(L^*))$.

Let $\boldsymbol{x} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L^*)) \cap \mathrm{Pref}(\mathrm{Vect}(L^*))$.

Note that $\mathrm{Pref}(\mathrm{Vect}(L^*)) = \mathrm{Vect}(L^*)\mathrm{Pref}(\mathrm{Vect}(L))$.

Let $\boldsymbol{x} = \boldsymbol{yz}$, where $\boldsymbol{y} \in \mathrm{Vect}(L^*)$ and $\boldsymbol{z} \in \mathrm{Pref}(\mathrm{Vect}(L))$.

Since $\mathrm{Vect}(L^*) \subseteq \overline{\mathrm{Vect}}(L^*)$, we have $\boldsymbol{y} \in \overline{\mathrm{Vect}}(L^*)$.

Let us consider $\boldsymbol{xa} = \boldsymbol{yza} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L^*))$. By the definition we have

$$(\forall i = 1, \ldots, n) \quad h_i(yza) = h_i(y)h_i(za) \in h_i(\mathrm{Pref}(L^*)) = h_i(L^*)h_i(\mathrm{Pref}(L)).$$

But this means that $(\forall i = 1, \ldots, n)$ $h_i(za) \in h_i(\mathrm{Pref}(L))$, so $\boldsymbol{za} \in \overline{\mathrm{Vect}}(\mathrm{Pref}(L))$. Since

$\overline{\text{Vect}}(\text{Pref}(L)) = \text{Pref}(\text{Vect}(L))$, we have $za \in \text{Pref}(\text{Vect}(L))$. Thus

$$xa = yza \in \text{Vect}(L^*)\text{Pref}(\text{Vect}(L)) = \text{Pref}(\text{Vect}(L^*)). \quad \Box$$

Lemma 6.12. *Let R be an E^*-expression of the form $R = (R')^*$ and let $L = |R|$. Then,*

$$x \in L \Leftrightarrow x \in \text{Pref}(L)\ \&\ (\exists x_1, \ldots, x_k \in C_R)(\forall i = 1, \ldots, n)\ h_{A(x_i)}(x) \in \{x_i\}^*$$
$$\&\ A(x) = A(x_1) \cup \cdots \cup A(x_k).$$

Proof. The proof directly follows from the definition of C_R. If $x \in L$, then every cycle included in x must be closed. $\Box$

Lemma 6.13. *Let R be a regular expression and let $L = |R|$. Let $xyz \in L$, $y \in C_R$, $y' \notin C_R$, $y = y'y''$, and $y' \neq \varepsilon$, $y'' \neq \varepsilon$.*
Then, $xy'z \notin L$.

Proof. The proof follows from the definition of C_R. The string $xy'z \notin L$ because it contains the beginning of an open cycle. $\Box$

Lemma 6.14. *Let R be an E^*-expression and let $L = |R|$. Let* $\text{Pref}(\text{Vect}(L)) = \overline{\text{Vect}}(\text{Pref}(L))$. *Let $(\forall X \in \text{CD}_{R^*})\ (D \cap X \times X)^+ = X \times X$.*
Then, $\text{Vect}(L^*) = \overline{\text{Vect}}(L^*)$.

Proof. Since $L^* \subseteq \text{Pref}(L^*)$, by Corollary 6.1 we have $\overline{\text{Vect}}(L^*) \subseteq \overline{\text{Vect}}(\text{Pref}(L^*))$. From Lemma 6.11 it follows that $\text{Pref}(\text{Vect}(L^*)) = \overline{\text{Vect}}(\text{Pref}(L^*))$, so $\overline{\text{Vect}}(L^*) \subseteq \text{Pref}(\text{Vect}(L^*))$.

Let $x \in \overline{\text{Vect}}(L^*) - \text{Vect}(L^*)$. Thus, $x \in \text{Pref}(\text{Vect}(L^*)) - \text{Vect}(L^*)$. From Lemma 6.12 it follows that $(\exists x' \in C_{R^*})\ h_{A(x')}(x) \notin \{x'\}^*$. Let us denote $y = h_{A(x')}(x)$. Of course, $A(y) \subseteq A(x')$ and $y \in \text{Pref}(\{x'\}^*)$. For every symbol a and every string s, let $\#_a(s)$ denote the number of occurrences of a in s. For instance, $\#_a(abca) = 2$.

Since $y \notin \{x'\}^*$, we have $(\exists a, b \in A(y))\ \#_a(y) \neq \#_b(y)$.

Since $x \in \overline{\text{Vect}}(L^*)$, we have $(\forall i = 1, \ldots, n)\ h_i(y) \in h_i(h_{A(x')}(L^*)) = h_i(\{x'\}^*)$. Assume that $(\exists A_j)\ \{a, b\} \in A_j$.

But since $\#_a(y) \neq \#_b(y)$, we have $\#_a(h_j(y)) \neq \#_b(h_j(y))$, so $h_j(y) \notin h_j(\{x'\}^*)$—a discrepancy. Thus $(a, b) \in I$.

But this means that $(a, b) \notin (D \cap A(x') \times A(x'))^+$.

In this way we have proved that $(D \cap A(x') \times A(x'))^+ \neq A(x') \times A(x')$, where $A(x') \in \text{CD}_R$. $\Box$

Lemmas 6.11 and 6.14 prove the implication $\Leftarrow$ for $R = (R_1)^*$.

Lemma 6.15. *Let R be a regular expression and let $L = |R|$.*

Then,

$$\mathrm{Vect}(L)=\overline{\mathrm{Vect}}(L) \Rightarrow$$

$$\Rightarrow (\forall X \in \mathrm{CD}_R)(\forall Y \subseteq X)\ (Y \text{ is a maximal subset of } X \text{ such that } (D \cap Y \times Y)^+ = Y \times Y) \Rightarrow Y \in \mathrm{CD}_R.$$

Proof. Assume that $x' \in C_R$, $X = A(x')$, and $(D \cap X \times X)^+ \neq X \times X$. Note that the above assumption implies $\mathrm{card}(X) \geq 2$.

Let $xx'y \in \mathrm{Vect}(L)$. One can easily show that such a sequence vector always exists. Let $Y \subseteq X$ be a maximal subset of X satisfying the condition $(D \cap Y \times Y)^+ = Y \times Y$, and $Y \notin \mathrm{CD}_R$. Of course, $\mathrm{card}(Y) \geq 1$.

Let $x'' = h_Y(x')$. Note that, by the definition, $xx''y \in \overline{\mathrm{Vect}}(L)$, and, by Lemma 6.13, $xx''y \notin \mathrm{Vect}(L)$. □

Lemma 6.16. *If R is an E^*-expression then the conditions given below are equivalent:*

(1) $(\forall X \in \mathrm{CD}_R)(\forall Y \subseteq X)$ *(Y is a maximal subset of X such that $(D \cap Y \times Y)^+ = Y \times Y$) $\Rightarrow Y \in \mathrm{CD}_R$.*

(2) $(\forall X \in \mathrm{CD}_R)\ (D \cap X \times X)^+ = X \times X.$

Proof. The proof follows from the definition of CD_R. □

From the last lemma we obtain the implication $\Rightarrow$ for $R = (R_1)^*$. In this way we proved Theorem 6.2.

Note that the condition $E_R \cap I = \emptyset$ is associated with the operation "$\cup$" only, and the condition $(\forall X \in \mathrm{CD}_R)(D \cap X \times X)^+ = X \times X$ is only associated with the operation "$*$".

Because in Lemmas 6.11 and 6.15 we assume nothing about the form of R, they hold in the general case. Thus, we may formulate the following theorem.

Theorem 6.17. *Let R be a regular expression and let $L = |R|$.*

Then,

$$[\mathrm{Vect}(L)=\overline{\mathrm{Vect}}(L)\ \&\ \mathrm{Pref}(\mathrm{Vect}(L))=\overline{\mathrm{Vect}}(\mathrm{Pref}(L))] \Rightarrow$$

$$\Rightarrow [E_R \cap I = \emptyset\ \&$$

$$(\forall X \in \mathrm{CD}_R)(\forall Y \subseteq X)\ (Y \text{ is a maximal subset of } X \text{ such that } (D \cap Y \times Y)^+ = Y \times Y) \Rightarrow Y \in \mathrm{CD}_R].$$

Proof. The proof follows from Lemmas 6.11 and 6.15. □

Theorem 4.3 is a special case of Theorem 6.17 (for $R = (R')^*$). We are now going to prove Theorem 4.4. The proof will be based on the results of Theorem 6.2.

Let $P = P_1 \ldots P_n$ be a GR1*-path, and let $A = \mathrm{Alpha}(P)$, $A_i = \mathrm{Alpha}(P_i)$ for $i = 1, \ldots, n$.

Let $B=\{a \mid (\exists i\in\{1,\dots,n\})\ \mathrm{occ}_i(a)>1\}$. Since P is a GR1*-path, for every $a\in A$ there is at most one i such that $\mathrm{occ}_i(a)>1$. For every $a\in B$, let i_a denote a number such that $\mathrm{occ}_{i_a}(a)>1$. Let $a\in B$, and let $m_a=\mathrm{occ}_{i_a}(a)$.

Let P' denote the result of converting P according to the following rules:

"For every $a\in B$:

(1) replace the ith occurrence of a in P_{i_a} by $a\&i$,

(2) for every $i=1,\dots,i_a-1, i_a+1,\dots,n$, replace an occurrence of a in P_i by the string $a\&1, a\&2,\dots, a\&m_a$."

The path P' is said to be a GE*-*representation* of the GR1*-path P. The above construction is essentially the same as the general transformation of generalized paths into GE*-paths given in [13]. Because P is a GR1*-path, a new numeration of repeated actions may be somewhat simpler than that of [13].

Example 6.18. Let P be the following GR1*-path:

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

In this case, $B=\{a, c\}$ and

$P'=$ **system**
  path $a\&1$; b, $a\&2$ **end**
  path $a\&1$, $a\&2$; $c\&1$; $c\&2$; $c\&3$ **end**
  path b, $a\&1$, $a\&2$; $c\&1$, $c\&2$, $c\&3$ **end**
endsystem.

Let $A'=\mathrm{Alpha}(P')$, $C=A-B$. Note that $C\subseteq A'$ and $A'-C\subseteq\{a\&i \mid a\in B\ \&\ i\in\{1,2,\dots\}\}$.

Let $h_\&:\mathrm{Vect}((A')^*)\to\mathrm{Vect}(A^*)$ be the following homomorphism:

$$(\forall b\in A')\quad h_\&(b)=\begin{cases} b & b\in C,\\ a & b=a\&i\in A'-C.\end{cases}$$

Lemma 6.19 (follows from [13]).

(1) $\mathrm{VFS}(P)=h_\&(\mathrm{VFS}(P'))$.

(2) $\mathrm{VFFS}(P)=h_\&(\mathrm{VFFS}(P'))$.

Proof (the idea). This is a consequence of the construction of P'. It turns out that Petri nets simulating P and P' (according to standard rules from [13, 14]) are isomorphic. Let us consider the following simple example.

Let

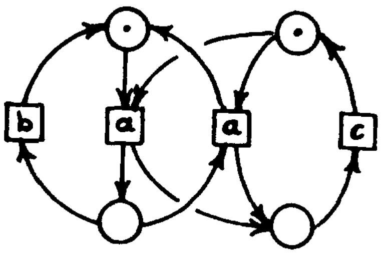
$P =$ **system path** a ; b, a **end path** a ; c **end endsystem.**

Thus

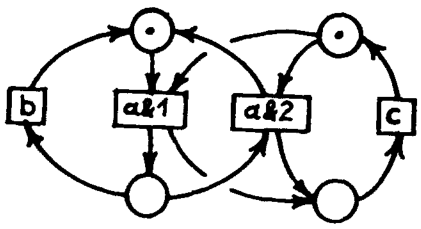
$P' =$ **system path** $a\&1$; b, $a\&2$ **end path** $a\&1$, $a\&2$; c **end endsystem.**

The appropriate simulating Petri nets $N(P)$ and $N(P')$ are the following:

N(P) =



N(P′) =



For more details, the reader is referred to [13, 14]. □

Let P_S be a single path, and let resource$(P_S) = \{x_1, \ldots, x_n\}$. Assume that $A =$ Alpha(P), $A_i = \hat{r}(x_i)$ for $i = 1, \ldots, n$. Let $P_C = P_S/x_1 \ldots P_S/x_n$. Assume that P_C is a GR1*-path.

Let PE$_S$ denote the result of converting P_S according to the following rule:

"For every $a \in A$, if a occurs more than once in P_S, then the ith occurrence of a is replaced by $a\&i$."

For instance, if $P_S =$ **path** a ; $(b, a)^*$, b ; c, a **end**, then PE$_S =$ **path** $a\&1$; $(b\&1, a\&2)^*$, $b\&2$; c ; $a\&3$ **end.**

Let us extend the resource association function r on Alpha(P') in the following way: $(\forall a\&i \in \text{Alpha}(P'))\ r(a\&i) = r(a)$.

Let PE$_C$ = PE$_S/x_1, \ldots,$ PE$_S/x_n$.

Lemma 6.20. PE$_C$ *is a* GE*-*representation of* P_C.

Proof. This is a simple consequence of the construction of the GE*-representation. □

Theorem 6.21. *Let* $L = |P_S|$.

Then:

$$(E_{P_S} \cap I = \emptyset \ \& \ (\forall X \in \text{CD}_{P_S})\ (D \cap X \times X)^+ = X \times X) \Rightarrow$$
$$\Rightarrow (\text{Vect}(L) = \overline{\text{Vect}}(L) \ \& \ \text{Pref}(\text{Vect}(L)) = \overline{\text{Vect}}(\text{Pref}(L))).$$

Proof. Assume that $E_{P_S} \cap I = \emptyset \;\&\; (\forall X \in \mathrm{CD}_{P_S})\,(D \cap X \times X)^+ = X \times X$.

Let $A' = \mathrm{Alpha}(\mathrm{PE}_S)$ and let $I_\&, D_\& \subseteq A' \times A'$ be the following relations:

$$I_\& = \{(\alpha, \beta) \mid (h_\&(\alpha), h_\&(\beta)) \in I\},$$

$$D_\& = A' \times A' - I_\&.$$

Let $L' = |\mathrm{PE}_S|$. From the definitions of PE_S and $h_\&$ we have

$$\mathrm{Vect}(L) = h_\&(\mathrm{Vect}(L')), \qquad \mathrm{Pref}(\mathrm{Vect}(L)) = h_\&(\mathrm{Pref}(\mathrm{Vect}(L'))).$$

From the above statement and the definitions of $I_\&$, $D_\&$ we obtain

(1) $(h_\&(\alpha), h_\&(\beta)) \in E_{P_S} \Rightarrow (\alpha, \beta) \in E_{\mathrm{PE}_S}$,

(2) $((\forall X \in \mathrm{CD}_{P_S})\,(D \cap X \times X)^+ = X \times X) \Rightarrow ((\forall X \in \mathrm{CD}_{\mathrm{PE}_S})\,(D_\& \cap X \times X)^+ = X \times X)$.

But this means simply that

$$(E_{P_S} \cap I = \emptyset \;\&\; (\forall X \in \mathrm{CD}_{P_S})\,(D \cap X \times X)^+ = X \times X) \Rightarrow$$
$$\Rightarrow (E_{\mathrm{PS}_S} \cap I = \emptyset \;\&\; (\forall X \in \mathrm{CD}_{\mathrm{PE}_S})\,(D_\& \cap X \times X)^+ = X \times X).$$

By Theorem 6.2 we have

$$(E_{\mathrm{PE}_S} \cap I = \emptyset \;\&\; (\forall X \in \mathrm{CD}_{\mathrm{PE}_S})\,(D_\& \cap X \times X)^+ = X \times X) \Leftrightarrow$$
$$\Leftrightarrow (\mathrm{Vect}(L') = \overline{\mathrm{Vect}}(L') \;\&\; \mathrm{Pref}(\mathrm{Vect}(L')) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L'))).$$

By the definition of $h_\&$ we can write

$$(\mathrm{Vect}(L') = \overline{\mathrm{Vect}}(L') \;\&\; \mathrm{Pref}(\mathrm{Vect}(L')) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L'))) \Rightarrow$$
$$\Rightarrow (h_\&(\mathrm{Vect}(L')) = h_\&(\overline{\mathrm{Vect}}(L')) \;\&\; h_\&(\mathrm{Pref}(\mathrm{Vect}(L')))$$
$$= h_\&(\overline{\mathrm{Vect}}(\mathrm{Pref}(L')))).$$

As we have stated above, from the definitions of PE_S and $h_\&$ we have

$$h_\&(\mathrm{Vect}(L')) = \mathrm{Vect}(L), \qquad h_\&(\mathrm{Pref}(\mathrm{Vect}(L'))) = \mathrm{Pref}(\mathrm{Vect}(L)).$$

From Lemmas 6.19 and 6.20 we obtain

$$h_\&(\overline{\mathrm{Vect}}(L')) = h_\&(\mathrm{VFFS}(P')) = \mathrm{VFFS}(P) = \overline{\mathrm{Vect}}(L),$$

$$h_\&(\overline{\mathrm{Vect}}(\mathrm{Pref}(L'))) = h_\&(\mathrm{VFS}(P')) = \mathrm{VFS}(P) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L)).$$

But this means that

$$(\mathrm{Vect}(L') = \overline{\mathrm{Vect}}(L') \;\&\; \mathrm{Pref}(\mathrm{Vect}(L')) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L'))) \Rightarrow$$
$$\Rightarrow (\mathrm{Vect}(L) = \overline{\mathrm{Vect}}(L) \;\&\; \mathrm{Pref}(\mathrm{Vect}(L)) = \overline{\mathrm{Vect}}(\mathrm{Pref}(L))),$$

which ends the proof of the theorem. □

Theorem 4.4 is a consequence of the above theorem. Necessary and sufficient conditions for the general case are an open problem still.

7. Final comments

The method presented above has two disadvantages: first, sometimes it leads to functionally different specifications, and second, the necessary and sufficient conditions for functional equivalence are not easy to verify, particularly the construction of the relation E may be uphill; furthermore, sufficient condition are unknown in the general case.

The second fault may be mended in future, but the first unfortunately not. The good point of the method lies in the fact that we start with a sequential solution. Long before now, people have stated that it is very difficult to comprehend the combined effect of activities which evolve simultaneously and with independent speeds. Up till now, the human imagination, not technology, is a main obstacle in use of concurrency in computers. It is hard to avoid the conclusion that we understand concurrent events by looking at sequential subsets of them. We suppose there are two natural methods of specifying concurrent systems. The first of them, very popular, consists in the logical decomposition of the problem into sequential in the course of nature components, independent designing each component, and next superposing all components. Among others, the COSY path expressions and Hoare's CSP [4] are examples of that approach. The second method is presented in [6, 7, 17, 16] and in this paper. For some applications, this second method seems to be more convenient (see examples in [17]). We also feel this paper can only be treated as a *first* step towards a methodology which starts with a primary sequential solution. The general transformations are probably more complicated than those presented in Section 2.

Acknowledgment

The author would like to thank Peter Lauer for his suggestions, comments and criticism, and for the invitation to Newcastle-upon-Tyne. The author is also indebted to the anonymous referee(s) for careful reading, helpful comments and finding errors. The work reported in this paper was partly supported by a grant from the Science and Engineering Research Council of Great Britain.

References

[1] P.J. Courtois, F. Heymans and D.L. Parnas, Concurrent control with "readers" and "writers", *CACM* **14** (10) (1971) 667-668.

[2] E.W. Dijkstra, Hierarchical ordering of sequential processes, in: C.A.R. Hoare and R.H. Perrott, eds., *Operating System Techniques* (Academic Press, New York, 1973).

[3] M.A. Harrison, *Introduction to Switching and Automata Theory* (McGraw-Hill, New York, 1965).

[4] C.A.R. Hoare, Communicating sequential processes, in: McKeag and McNaughton, eds., *On the Construction of Programs* (Cambridge University Press, London, 1980).

[5] R. Janicki, A characterisation of concurrency-like relations, *Lecture Notes in Computer Science* **70** (Springer, Berlin, 1979) 320-333.

[6] R. Janicki, On the design of concurrent systems, *Proc. 2nd conf. on Distributed Computing Systems*, Paris, 1981 (IEEE Press, New York, 1981) 455-466.

[7] R. Janicki, A construction of concurrent schemes by means of sequential solution and concurrency relations, *Lecture Notes in Computer Science* **107** (Springer, Berlin, 1981) 327-334.

[8] R. Janicki, Nets, sequential components and concurrency relations, *Theoret. Comput. Sci.* **29** (1984) 87-121.

[9] R. Janicki, A method for developing concurrent systems, *Lecture Notes in Computer Science* **167** (Springer, Berlin, 1984) 155-166.

[10] P.E. Lauer, Synchronization of concurrent processes without globality assumptions, in: K.G. Beauchamp, ed., *New Advances in Distributed Computer Systems*, NATO Advanced Study Institutes Series (Reidel, Dordrecht, 1982) 341-366.

[11] P.E. Lauer and R.H. Campbell, Formal semantics for a class of high level primitives for coordinating concurerent processes, *Acta Informatica* **5** (1975) 247-322.

[12] P.E. Lauer and M.W. Shields, Abstract specification of resource accessing disciplines: adequacy, starvation, priority and interrupts, *SIGPLAN Notices* **13** (12) (1978) 41-58.

[13] P.E. Lauer, M.W. Shields and E. Best, Formal theory of the basic COSY notation, TR 143, Comput. Lab., Univ. of Newcastle-upon-Tyne, 1979.

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

[15] P.E. Lauer, P.R. Torrigiani and M.W. Shields, COSY: A system specification based on paths and processes, *Acta Informatica* **12** (1979) 109-158.

[16] C. Lengauer, A methodology for programming with concurrency: The formalism, *Sci. Comput. Programm.* **2** (1982) 19-52.

[17] C. Lengauer and E.C.R. Hehner, A methodology for programming with concurrency: An informal presentation, *Sci. Comput. Programm.* **2** (1982) 1-18.

[18] A. Mazurkiewicz, Concurrent program schemes and their interpretations, DAIMI PB-78, Aarhus Univ. Press, 1977.

[19] S.S. Patil, Limitations and capabilities of Dijkstra's semaphore primitives for co-ordination among processes, Project MAC, Computation Structures Group Memo 57, 1971.

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

[21] M.W. Shields, Is COSY big enough? Notes towards a study of deterministic concurrent machines, Report ASM/77, Comput. Lab., Univ. of Newcastle-upon-Tyne, 1980.

[22] M.W. Shields, On the non-sequential behaviour of systems possessing a generalized free-choice property, Internal Report CRS-91-81, Dept. of Comput. Sci., Univ. of Edinburgh, 1981.

[23] M.W. Shields and P.E. Lauer, A semantics for concurrent systems, *Lecture Notes in Computer Science* **71** (Springer, Berlin, 1979).