4

True versus artificial concurrency

A. Mazurkiewicz[1]
Institute of Computer Science of PAS
Ordona 21, 01-237 Warsaw, Poland
tel.+48(022)362885, fax: +48(022)376564
e-mail: amaz@ipipan.waw.pl

Abstract
Some theoretical issues related to concurrent processed are discussed: formal description of concurrent systems, concurrent systems synchronization, refinement of systems, fairness assumption. These issues are considered on possibly high level of abstraction enabling their proper identification, formulation, and formalization. So-called prefix functions serve as a model of concurrent processes. The notion of prefix functions is a unifying concept for description purposes of a number of existing representations of concurrent processes based on finite set of atomic actions.

Keywords
Concurrency, processes, partial order, synchronization, fairness, refinement.

1 INTRODUCTION

Several theoretical issues came up while proving correctness of concurrent systems. The main source of difficulties is confusion created by interleaving approach to concurrency, hiding independence of actions by nondeterminism of their execution. Some of these difficulties can be avoided by making independence of actions explicit and use it in a properly chosen theoretical framework. The aim of this outline is to present some of issues involving such an explicit reference to concurrency and to show – using rather high abstraction level – a way and means to cope with them.

The present outline addresses some issues inherently involving concurrency: synchronization of systems, introducing concurrency by collecting together systems (even sequential), actions refinement, where independent actions should have independent refinements, and fairness, where concurrency must be taken into account for an adequate definition of the notion. The event based approach is applied, as e.g. in the event structures (Winskel 1988); but here all events are occurrences of actions taken from a finite repertoire (alphabet).

First, to create some useful theoretical tools, the notion of *prefix function* is defined. The defined formalism is illustrated by some examples of simple and known systems.

[1]Partially supported by grant 8 T11C 029 08 of KBN (Scientific Research Counsil of Poland)

Behaviour of the introduced systems is presented in different ways, enabling comparison of their descriptive possibilities. Using the introduced formalism, the synchronization of systems is defined and illustrated by an example taken from the theory of Petri nets. Next, the problem of action refinement is discussed and related to the properties of traces. Finally, issues concerning inevitable properties of systems are taken into consideration. The main topic of this issue is a proper definition of 'observations' of the system run; observation is an increasing set of system histories such that any history of a single system run is an initial part of a sufficiently extensive history of the observation. A property of system states is inevitable, if any system observation encounters, sooner or later, a state with this property. Thus, for proving inevitability of a property of system states, one has to prove that fairness of an observation guarantees nonemptiness of its intersection with the set of states meeting the property in question.

2 BASIC NOTIONS

Let S be a set. A binary relation $\leq \subseteq S \times S$ is an *ordering* in S, if it is reflexive, transitive, and antisymmetric. The pair $(S, \leq)$ where S is a set and $\leq$ is an ordering in S is called a ***partially ordered set***, abbreviated to ***poset***. If the ordering relation is known, poset $(S, \leq)$ is usually identified with S. If $x \leq y$, y is said to ***dominate*** x, or x to ***be dominated by*** y. Two elements of S are *consistent*, if both of them are dominated by a common element of S. Two elements of S are ***comparable***, if one of them is dominated by the other. Clearly, comparable elements are consistent. A subset of S is *linear*, if all its elements are comparable. For any poset $(S, \leq)$ and any $Q \subseteq S$, Q is ordered by the restriction of $\leq$ to Q, (the relation $\leq \cap (Q \times Q)$); the poset $(Q, \leq \cap (Q \times Q))$ is called a ***subposet*** of S. A subset Q of S is *cofinal* with S, if for each element of S is dominated by a suitable element of Q. In graphical representations of partial orderings arcs resulting by transitivity from others are usually omitted.

Any finite set will be called here an *alphabet* and its elements *symbols*. If Σ is an alphabet, Σ^* is the set of all finite sequences of symbols from Σ, including the empty sequence ϵ, and $\Sigma^+ = \Sigma^* - \{\epsilon\}$. Sequences in Σ^* are called *strings* over Σ. The number of occurrences of symbol a in string u is denoted by $u(a)$. Concatenation of string u with string v is denoted by uv. Subsets of Σ^* are called *languages* over Σ. String u is a ***prefix*** of string w, if there exists a string v with $uv = w$. Any language is ordered by the prefix relation: $u \leq v$ if u is a prefix of v. The set of all prefixes of a string u is denoted by $\mathbf{P}(u)$; the set of all prefixes of all strings in language L is denoted by $\mathbf{P}(L)$. Clearly, $L \subseteq \mathbf{P}(L)$ for any language L. A language L is *prefix closed*, if $\mathbf{P}(L) = L$. From above definitions it follows that each nonempty prefix closed language contains the empty string. The greatest prefix closed subset of language L is denoted by $\ker(L)$ (such a language always exists and is unique).

Let I be a set (of indices); a function which to each $i \in I$ assigns a set X_i is called a *family* of sets and is denoted by $\{X_i\}_{i \in I}$. Let $\{\Sigma_i\}_{i \in I}$ be a family of alphabets; for any $w \in (\bigcup_{i \in I} \Sigma_i)^*$ and any $i \in I$ the string arising from w by erasing all symbols not in Σ_i is called the *projection* of w onto Σ_i and is denoted by $\pi_i(w)$.

A mapping $\phi : X \longrightarrow Y$ is an injection, if $\phi(x') = \phi(x'') \Rightarrow x' = x''$ and it is a surjection, if it is for each $y \in Y$ there is $x \in X$ such that $\phi(x) = y$. A mapping is a ***bijection***, if it is an injection as well as a surjection. If $\phi_1 : X \longrightarrow Y$ is a surjection,ϕ_2 :

$Y \longrightarrow Z$ is any mapping, then $\phi_1 \circ \phi_2$ is the mapping $\phi : X \longrightarrow Z$ defined by $\phi(x) = \phi_2(\phi_1(x))$.

A binary relation $\psi \subseteq X \times Y$ is a partial function from X to Y if $x\psi y', x\psi y''$ implies $y' = y''$. To denote partial functions from X to Y the notation $\psi : X \longrightarrow Y$ is used, if partiality of ψ is understood from the context. For any partial function $\psi : X \longrightarrow Y$ the set $\{x \in X \mid \exists y \in Y : \psi(x) = y\}$ is the *domain* of ψ, denoted by $\mathrm{Dom}(\psi)$, and the set $\{y \in Y \mid \exists x \in X : \psi(x) = y\}$ is the *range* of ψ, denoted by $\mathrm{Rng}(\psi)$. If ϕ, ψ are partial functions the equality $\phi(x) = \psi(x)$ means that either both are defined for x and their values for x are equal, or both of them are undefined for x.

3 PREFIX FUNCTIONS

Let Σ be an alphabet. Any partial function defined on Σ^* with a prefix closed domain will be called here a *prefix function* over Σ, abbreviated sometimes to a *function* over Σ, if its prefix closedness is understood. The domain of a prefix function is its *language* and its range is its set of *states*. Any superset of the range of a prefix function forms its *state space*. Two prefix functions σ_1, σ_2 are *isomorphic*, in symbols $\sigma_1 \simeq \sigma_2$, if their alphabets are identical and there exists a bijection ϕ between their ranges such that

$$\sigma_1 \circ \phi = \sigma_2.$$

Isomorphic prefix function are considered as identical; it means that prefix function are in fact classes of isomorphic functions with prefix closed domains. From this it follows that each prefix function σ over Σ has its *canonical form* with 2^{Σ^*} as its state space:

$$\sigma(w) = \{u \in \mathrm{Dom}(\sigma) \mid \sigma(u) = \sigma(w)\}$$

for each string w in the domain of σ. To each prefix function corresponds an equivalence relation $\equiv_\sigma$ in its domain called the *natural equivalence* of σ, such that $u \equiv_\sigma v \Leftrightarrow \sigma(u) = \sigma(v)$. Equivalence classes of $\equiv_\sigma$ are denoted as usual: $[w]_\sigma$ is the equivalence class of $\equiv_\sigma$ containing string w.

Prefix functions can be viewed as a tool for the discrete systems behaviour description, interpretating their arguments as the system actions sequences and their values as the resulting states. The function assigning to each (initiated) transition sequence of a Petri net the resulting marking is an example of a prefix function. Another example is related to transition systems with a fixed initial state: a function, assigning to each sequence of transitions its resulting state is a prefix function.

For any prefix function σ over Σ and each $a \in \Sigma$ let the *transition* relation of σ be defined as follows:

$$s' \overset{a}{\rightarrow}_\sigma s'' \Leftrightarrow \exists u \in \Sigma^* : s' = \sigma(u), s'' = \sigma(ua).$$

for each $s', s'' \in \mathrm{Rng}(\sigma)$ and $a \in \Sigma$. The *step* relation of σ is the relation $\rightarrow_\sigma$ defined as

$$s' \rightarrow_\sigma s'' \Leftrightarrow \exists a \in \Sigma : s' \overset{a}{\rightarrow}_\sigma s''$$

and the transitive and reflexive closure of $\rightarrow_\sigma$ is the *progress* relation of σ. The subscript σ is omitted if it causes no ambiguity. Graphical representation of the transition relation of a prefix function is the *diagram* of this function.

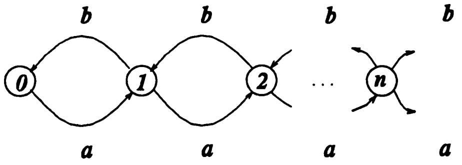

Figure 1 The diagram of prefix function σ_0 from Example 1.

Example 1. Let $L = \ker\{w \mid w(a) \geq w(b)\}$. Thus, L is a prefix closed language with strings containing not more occurrences of b than of a. Then $\sigma_0 : L \longrightarrow \mathbf{N}$ such that $\sigma_0(w) = w(a) - w(b)$ for all $w \in L$ is a prefix function. The diagram of σ_0 is shown in Figure 1. □

A prefix function is *monotonous*, if its progress relation is an ordering; in this case the symbol $\leq$ rather than $\rightarrow^*$ is used. A monotonous prefix function is *strongly monotonous*, if $s' \rightarrow s'' \Rightarrow s' \neq s''$. Monotonous prefix functions are of primary interest in the present outline, since they can serve as a tool for converting sequences of actions of concurrent system into its histories that can be ordered in a different way. In concurrent approach some different sequences of actions may be only different descriptions of the same system run; by collecting them it is possible to recover the 'true' history of the system behaviour, ordered by the casual relation. In such cases the values of a monotonous prefix function can be interpreted as system (initial) histories rather than system states. The assumed monotonicity reflects the fact that the system histories can only grow, hence never can be repeated in the system execution.

The formalism introduced by prefix function allows us to make a clear distinction between sequential and nonsequential behaviours.

Example 2. Consider two discrete systems executing actions a and b. The first one performs action a, then action b, and halts, or action b, then action a, and halts. The choice between these two possibilities of acting is random. The second performs concurrently actions a and b and halts. The difference between the two systems becomes clear by describing them by two different prefix functions, σ' and σ''. Both function have the same alphabet, namely $\{a, b\}$, and the same domain, namely $\mathbf{P}\{ab, ba\}$; but σ' is the identity function, while the canonical representation of σ'' is given by equalities $\sigma''(\epsilon) = \{\epsilon\}, \sigma''(a) = \{a\}, \sigma''(b) = \{b\}$ and $\sigma''(ab) = \sigma''(ba) = \{ab, ba\}$, identyfying in this way executions ab and ba. Both prefix functions are monotonous. Clearly, σ' is not isomorphic with σ''; the above description makes explicit the difference between choice and concurrency. However, for special purposes both models can be equally useful; the choice between them depend exclusively upon the aims of formalization. □

The above example suggests calling *sequential* those prefix functions that are isomorphic to identity functions, and as *nonsequential* the others. The state ordering of sequential prefix functions is tree-like; nonsequentiality of a monotonous prefix function is related to the existence in its range some states incomparable but consistent. Interpreting values of a prefix function as initial histories and the state ordering as their inclusion, incomparable but consistent values of prefix function correspond to incomparable initial parts of the same history; incomparable and inconsistent values correspond to initial parts of different histories. Thus, histories in the range of sequential prefix functions are either linearly ordered, then they are initial parts of the same global history, or incomparable, then they are initial parts of different histories. In the range of a nonsequential prefix function two incomparable histories can be initial parts of the same global history, then they are consistent, or can be initial parts of some different global histories, then they are parts of different histories.

If actions creating the behaviour of a discrete system occur in a linear order, its partial histories are ordered linearly too. If some actions occur independently of each other, their occurrences in the behaviour are not comparable but they contribute to create in some future a common history. Existence of noncomparable but consistent histories of a system is an evidence of concurrent execution of some system actions.

In the prefix function formalism independent execution of actions can be expressed by assigning to different sequences of action symbols a common value, representing common history composed by their independent executions. The natural equivalence of a prefix function identifies different segmentation of the same global history into its initial subhistories.

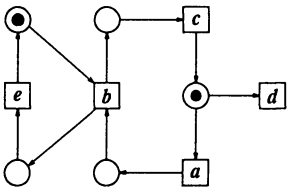

Figure 2 An elementary net system.

Let Σ be an alphabet and Dep $\subseteq \Sigma^2$ be a symmetric and reflexive relation called *dependency* in Σ and Ind $= \Sigma^2 -$ Dep *independency* in Σ. An alphabet together with a dependency (or independency) relation is a *concurrent alphabet.* Let $\equiv$ be the least congruence in Σ^* such that $ab \equiv ba$ for all $(a, b) \in$ Ind (i.e. transitive and reflexive closure of the relation

$$\{(uabv, ubav) \mid u, v \in \Sigma^*, (a, b) \in \text{Ind}\}.$$

This equivalence is the *trace equivalence* in Σ w.r. to Ind (or Dep) and equivalence classes of $\equiv$ are called *traces* over (Σ, Ind) (or (Σ, Dep)) (Mazurkiewicz, 1977). A trace containing string w is, as usual, denoted by $[w]$; the set of all traces with representants in $L \subseteq \Sigma^*$ is denoted by $[L]$; in particular, $[\Sigma^*]$ denotes the set of all traces over Σ. If L is a prefix closed language over Σ, then the function defined by $\sigma(w) = [w]$ for all $w \in [L]$ is a prefix function. The canonical form of this function is given by equalities $\sigma(w) = [w] \cap L$ and $\equiv \cap L^2$ is the natural equivalence of σ. Prefix functions with traces as values will be called *trace prefix functions.*

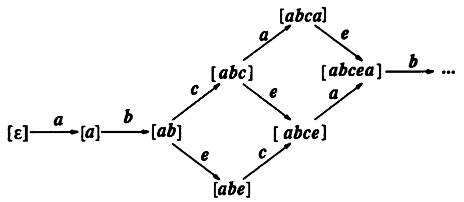

Figure 3 The diagram of the prefix function from Example 3.

In fact, an attempt to generalize the trace concept was the primary motivation for introducing prefix functions. The traces, as defined above, turned out to be an adequate tool for behaviour descriptions of so-called elementary net systems, (Rozenberg, 1987 and Thiagarajan, 1987) being a restricted version of Petri nets (Petri, 1976). More information about traces can be found in (Diekert, Rozenberg, 1994).

Example 3. Consider elementary net system represented graphically in Figure 2. From the theory of such systems it follows that the dependence relation of this system is Dep $= \{a,b,c,d\}^2 \cup \{b,e\}^2$; hence, Ind $= \{(a,e),(c,e),(d,e),(e,a),(e,c),(e,d)\}$. Thus, the trace equivalence contains pairs: $ae \equiv ea, ce \equiv ec, de \equiv ed$. The behaviour of this system can be described by prefix function σ over alphabet $\{a,b,c,d,e\}$, with $\sigma(u) = [u]$, Dom$(\sigma) =$ $\mathbf{P}\{w \mid \exists u \in (abce)^*d : w \equiv u\}$. The initial part of the diagram of σ is given in Figure 3. □

Let Σ be an alphabet and let $\Sigma^\oplus$ denote the set of all mappings from Σ to the set of all nonnegative integers. Elements of $\Sigma^\oplus$ are called *multisets* over Σ. Multisets over Σ can be viewed as the elements of the free commutative monoid $(\Sigma^\oplus, +, 0)$ generated by Σ, with $+$ as the (totally commutative) monoid operation, called *addition*, and 0 as the neutral element). Multisets over Σ can be represented by linear forms

$$k_1a_1 + k_2a_2 + \cdots + k_na_n,$$

where $n \geq 0$ (if $n = 0$ the above form is reduced to 0), $k_1, k_2, \ldots, k_n$ are nonnegative integers and $a_1, a_2, \ldots, a_n$ are members of Σ. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ and let $\mu : \Sigma^* \longrightarrow \Sigma^\oplus$ be mapping such that $\mu(w) = k_1a_1 + k_2a_2 + \cdots + k_na_n$, where $k_i = w(a_i)$ for each $i, 1 \leq i \leq n$, i.e. mapping converting strings into multisets over the same alphabet (recall that $w(a_i)$ denotes the number of occurrences of symbol a_i in string w). Prefix function σ such that $\sigma(w) = \mu(w)$ for each w in its domain is called a *multiset prefix function.*

Example 4. Now, let us consider four different description of the same discerete process, namely the behaviour of the so-called producer – consumer system, consisting of two agents; one of them is producing objects and putting them into a common store, the second is consuming stored objects. Both actions are independent of each other, with only one obvious restriction: if the store is empty, the consuming agent must wait until some objects are supplied by the producer. All processes has then the same alphabet $\{a,b\}$ of actions: a denotes production, b consuming. The domain of all prefix functions discussed here is the set D, defined already in Example 1:

$$D = \ker\{w \mid w(a) \geq w(b)\}.$$

(a prefix closed language with all strings containing not more occurrences of b than of a).

The first prefix function, σ_0 has been already defined in Example 1. The second, σ_1, is the identity function: $\sigma_1(u) = u$ for all $u \in D$. Then states are simply execution sequences, induced equivalence is the identity relation, equivalence classes are singletons. The ordering of σ_1 is presented in Figure 4.

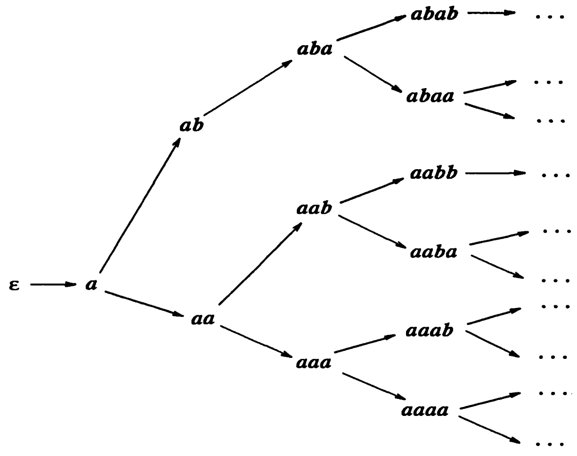

Figure 4 State ordering of σ_1.

The third prefix function is defined by the function σ_2 where $\sigma_2(u) = \mu(u)$ for all $u \in D$ (recall that μ is the function which for a given strings returns corresponding it multiset). The state ordering of σ_2 is componentwise:

$$\sigma_2(u) \leq \sigma_2(v) \Leftrightarrow u(a) \leq v(a), u(b) \leq v(b).$$

The natural equivalence of σ_2 identifies all strings with the same number of symbol occurrences, independently of their arrangement within strings; they are permutation classes of strings in D. The state ordering of σ_2 is given in Figure 5.

The last one is σ_3; its value for any $u \in D$ with $u(b) = m, u(a) = n (n, m \geq 0)$, is a node labelled graph with nodes $a_1, a_2, \ldots, a_n$ labelled with a, nodes $b_1, b_2, \ldots, b_m$ labelled with b, and the following arcs:

$$\{(a_i, a_{i+1}) \mid 1 \leq i < n\} \cup \{(a_j, b_j) \mid 1 \leq j \leq m\} \cup \{(b_j, b_{j+1}) \mid 1 \leq j < m\}.$$

States of σ_3 are node labelled graphs expressing explicitely causal relationship between producing and consuming actions. The state ordering of σ_3 is the ordering of the set of

$$\begin{tikzcd}
 & & & & & & \\
 & & & & 2a+2b \arrow[r] & 3a+2b \arrow[r, dash] \arrow[ur, dash] & \\
 & & a+b \arrow[r] & 2a+b \arrow[r] \arrow[ur] & 3a+b \arrow[r] \arrow[ur] & 4a+b \arrow[r, dash] \arrow[ur, dash] & \\
0 \arrow[r] & a \arrow[r] \arrow[ur] & 2a \arrow[r] \arrow[ur] & 3a \arrow[r] \arrow[ur] & 4a \arrow[r] \arrow[ur] & 5a \arrow[r, dash] \arrow[ur, dash] &
\end{tikzcd}$$

Figure 5 State ordering of σ_2.

$$\begin{tikzcd}
 & b \arrow[r] & b \arrow[r] & b \arrow[r] & b & \\
a \arrow[r] \arrow[ur] & a \arrow[r] \arrow[ur] & a \arrow[r] \arrow[ur] & a \arrow[r] \arrow[ur] & a \arrow[r] & a
\end{tikzcd}$$

Figure 6 The state $\sigma_3(aababa)$ (corresponding to multiset $4a + 2b$).

oriented acyclic graphs by the relation 'to be an initial full subgraph'. An example of a state in the range of σ_3 is given in Figure 6.

No two of prefix functions $\sigma_0, \sigma_1, \sigma_2$ are isomorphic. Functions σ_1, σ_2, and σ_3 are ordered, while σ_0 is not; the state ordering of σ_1 is a tree-like ordering, while those of σ_2 and σ_3 are not. Equivalence classes of σ_1 are singletons; those of σ_0 contain infinitely many strings, and those of σ_2 and σ_3 contain only finite number of elements. Functions σ_2 and σ_3 are isomorphic.

The first prefix function, σ_0 from Example 1, represents the point of view of a store manager. Sequences of producing and/or consuming are identified, if they result in the same number of items in the store.

The second function, σ_1, describes the producer – consumer activity from the viewpoint of sequential observers. Any of them, despite the existing independency of the agent actions, observes the system run as a sequence of producing and consuming actions. Different observers can see these sequences in different ways; the sequential order of action occurrences they notice does not reflect faithfully the causal ordering of agents actions.

The third prefix function, σ_2, is intended to describe the existing concurrency between producing and consuming actions explicitely. It turns out that accepting multisets over alphabet $\{a, b\}$ as the state descriptions the required concurrency is captured in a satisfactory way. It also turns out that the causal relationship between producing and consuming actions can be inferred from the ordering of states of σ_2. In this setup two execution sequences are identified, if they describe the same run of the system, with different ordering of actions caused by their mutual independence. Prefix function σ_2 is isomorphic with σ_3; the last one presents causal dependencies in the system as a graph. Values of σ_3 are graphs; in this graphs the transitive and reflexive closure of arc relation indicates the partial order of action occurrences. Whenever there is no causal relationship between producer and consumer actions, there is no chain of arcs joining their occurrences in the graph. In other words, σ_3 function uses labelled partial order (lpo) technique for describing concurrency; labelled partial orders are the same objects as pomsets of Pratt (1986). □

4 PREFIX FUNCTIONS SYNCHRONIZATION

One of the most important operations on discrete systems is their synchronization. Synchronization of systems corresponds to the synchronization of prefix functions defined below and can be used as a tool for combining simple systems into more complex ones. Let I be a finite set (of indices) and let $F = \{\sigma_i\}_{i \in I}$ be a family of prefix functions, $\sigma_i = (\Sigma_i)^* \longrightarrow S_i$ for $i \in I$. The *synchronization* of family σ is a prefix function σ over the union of alphabets Σ_i:

$$\sigma : (\bigcup_{i \in I} \Sigma_i)^* \longrightarrow \prod_{i \in I} S_i$$

with domain

$$\ker\{w \in \Sigma^* \mid \pi_i(w) \in \mathrm{Dom}\,(\sigma_i)\}$$

and with values being tuples of values of component functions:

$$(\sigma(w))_i = \sigma_i(\pi_i(w)) \qquad (i \in I),$$

where $(\sigma(w))_i$ denotes the i-th component of the tuple $\sigma(w)$ from the cartesian product. Notice the kernel operation in the definition of the domain of the synchronized family; its application guarantees the prefix closedness of the domain. The idea of the synchronization defined above has been used for modular description of Petri nets (Mazurkiewicz, 1985) and for creating string vectors of Shields (1979).

The synchronization of family $\{\sigma_i\}_{i \in I}$ will be denoted by $\|_{i \in I}\, \sigma_i$.

1. *The synchronization of any finite family of prefix functions is a prefix function.*

For two-element family of prefix functions write $\sigma_1 \parallel \sigma_2$ rather than $\|_{i \in \{1,2\}}\, \sigma_i$; thus, $\|$ can be viewed also as a binary operation on prefix functions.

2. *Synchronization operation is idempotent, commutative, and associative, i.e. for all prefix functions $\sigma, \sigma_1, \sigma_2, \sigma_3$:*

$$\begin{aligned} \sigma \parallel \sigma &= \sigma, \\ \sigma_1 \parallel \sigma_2 &= \sigma_2 \parallel \sigma_1, \\ (\sigma_1 \parallel \sigma_2) \parallel \sigma_3 &= \sigma_1 \parallel (\sigma_2 \parallel \sigma_3). \end{aligned}$$

Idempotency follows from the fact that the mapping which to each argument assigns its two identical copies is a bijection. Commutativity is clear; associativity follows from the associativity of product operation.

3. *Synchronization of monotonous prefix functions is monotonous.*

It follows from the definition of synchronization, since the ordering of the synchronized prefix functions is the product of its components orderings.

For any family of sequential prefix functions define an independence relation Ind in the union of the family alphabets as follows:

$$(a, b) \in \text{Ind} \Leftrightarrow \forall i, j \in I : a \in \Sigma_i, b \in \Sigma_j \Rightarrow i \neq j,$$

where I is the set of indices of the family and σ_i, Σ_j are alphabets of prefix functions of the family. Such a relation is called the independence *generated* by the family.

4. *Synchronization of any family of sequential prefix function is a prefix function with values being traces w.r. to the independence relation generated by the family.*

Therefore, synchronization can be viewed as a tool for introducing independency of some actions; this independency, however, is 'static', i.e. fixed for all possible runs of the described system. By the defined above synchronization it is not possible to introduce a 'context-sensitive' concurrency (depending upon the system history).

5. *Synchronization of any family of multiset prefix functions is a multiset prefix function.*

It follows from the fact that for all strings u, v over $\bigcup_{i\in I} \Sigma_i$ the following equivalence holds:

$$\mu(u) = \mu(v) \Leftrightarrow \forall i \in I : \mu(\pi_i(u)) = \mu(\pi_i(v)),$$

where $\pi_i(w)$ denotes the projection of w onto Σ_i.

Let I be a finite set of indices and let A_p, B_p be alphabets for each $p \in P$, with $A_p = \{a_1^p, a_2^p, \ldots, a_{n_p}^p\}, B_p = \{b_1^p, b_2^p, \ldots, b_{m_p}^p\}, n_p, m_p \geq 0, p \in P$. Let, for each $p \in P$ and $u \in (A_p \cup B_p)^*$,

$$\alpha_p(u) \Leftrightarrow m_p + u(a_1^p) + u(a_2^p) + \cdots + u(a_{n_p}^p) \geq u(b_1^p) + u(b_2^p) + \cdots + u(b_{m_p}^p). \qquad (1)$$

Let $\theta_p : (A_p \cup B_p)^* \longrightarrow (A_p \cup B_p)^{\oplus}$ be a multiset prefix function with the domain Dom_p:

$$\text{Dom}_p = \ker\{w \in (A_p \cup B_p)^* \mid \alpha_p(w)\}$$

Comparing the above definition with that from Example 2, θ_i leads to the conclusion that θ_i can be viewed as the prefix function description of the producer – consumer system with producers represented by elements of A_p and consumers represented by elements of B_p, with m_p as the initial contents of the store, for all $p \in P$. It can be viewed as well as a one-place Petri net N_p, with a single place p, initial marking m_p, input transitions A_p and output ones B_p. In Figure 7 an example of such net is given, with input transitions a_1, a_2, a_3, output transition b_1, b_2, and the initial marking 3. The behaviour of N_p is given by multiset prefix function θ_p.

Set now $A = \bigcup_{p\in P} A_p, B = \bigcup_{p\in P} B_p$ and consider Petri net (P, T, F, m^0) with unbounded capacity of places and 1-weighted arcs such that

$$T = A \cup B, \quad m^0(p) = m_p, \quad \text{for all } p \in P$$
$$F = \{(a, p) \mid a \in A_p, p \in P\} \cup \{(p, b) \mid b \in B_p, p \in P\}.$$

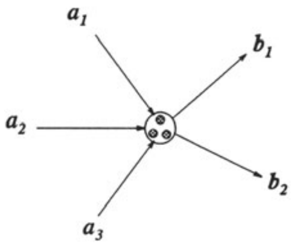

Figure 7 An atomic place/transition net.

The above net is constructed from atomic nets by identyfying some transitions of components, with places that form a vector indexed by elements of P, and with marking inherited from markings of atoms.The behaviour of the constructed net is the synchronization of behaviours of its components:

$$\Theta = \|_{i \in I} \, \theta_i.$$

It can be shown that Θ is a multiset prefix function with domain

$$\ker\{w \in (A \cup B)^* \mid \bigwedge_{p \in P} \alpha_p(w)\}$$

where α_p is the condition defined in (1) for each $p \in P$. Thus, the prefix function synchronization enables establishing a compositional semantics of Petri nets (Mazurkiewicz, 1985) and illustrates the use of synchronization technique in describing some complex concurrent systems.

5 PREFIX FUNCTIONS REFINEMENT

Intuitively, refinement of a discrete system consists in replacing an original system actions by some combination of other actions giving more accurate description of the system behaviour. A good candidate for a formal model of refinement is string substitution: a string of symbols is substituted for a single symbol in the behaviour description and this substitution is extended from symbols to strings and languages. In case of nonsequential systems the situation is not so simple.

Example 5. Let consider a system in which actions a and b are performed independently of each other, and next the system halts. The behaviour of such a system can be represented traditionally by the prefix language $\mathbf{P}\{ab, ba\}$. Suppose the system is refined by replacing action a by the sequence a_1a_2 of 'finer' actions. Then the system behaviour might be transformed in a straightforward way into $\mathbf{P}\{a_1a_2b, ba_1a_2\}$. But, according to intuitive understanding concurrency, such a refinement should lead to another behaviour, namely to $\mathbf{P}\{a_1a_2b, a_1ba_2, ba_1a_2\}$. In other words, refinement of an action should respect its independence of other actions, hence a_1 as well as a_2 should be independent of b. It is not the case while using sequential description of systems. □

The above example shows that the behaviour of refined system is not the same as the refinement of the original system behaviour; to transform properly the original behaviour into the refined one, additional information on mutual dependencies (or independencies) in the set of refined actions should be supplied.

It is worthwhile to note, however, that for sequential systems in which there are no independent actions, the refinement procedure as given above leads to satisfactory results. It suggest to combine refinement of sequential systems with synchronization operation for validation of more complex, nonsequential systems (see also: Abadi, Lamport 1989).

Prefix functions considered in this section are assumed to be monotonous. Let σ_1, σ_2 be prefix functions over Σ_1, Σ_2, respectively. *Refinement injection* of σ_1 into σ_2 is any monotone and cofinal injection of $\mathrm{Rng}\,(\sigma_1)$ into $\mathrm{Rng}\,(\sigma_2)$, i.e. any mapping $\phi : \mathrm{Rng}\,(\sigma_1) \longrightarrow \mathrm{Rng}\,(\sigma_2)$ such that

$$\forall s', s'' \in Rng(\sigma_1) : s' \leq s'' \Rightarrow \phi(s') \leq \phi(s''), \text{ and}$$
$$\forall s'' \in \mathrm{Rng}\,(\sigma_2) : \exists s' \in \mathrm{Rng}\,(\sigma_1) : s'' \leq (\phi(s')).$$

A prefix function σ_2 is a refinement of another prefix function σ_1 if there exists arefinement injection of σ_1 to σ_2.

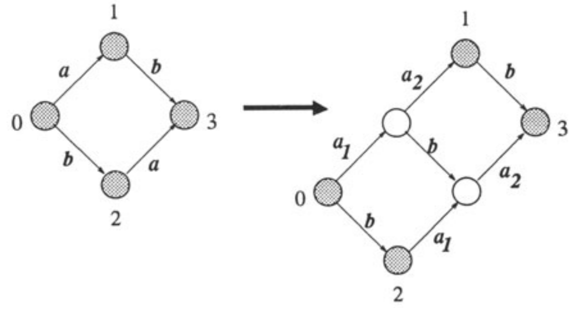

Figure 8 An example of refinement.

In Figure 8 a refinement of the prefix function defined in Example 5 is given. The states corresponding by injection to others are shadowed.

The intuition behind this definition is the following. The refined system has in general more actions and states than the original one. Anyhow, it should imitate the behaviour of the original system. It means that some states of the refined system must correspond to all states of the original one (hence injection). Moreover, they have to come out in the same order as corresponding to them states of the original system (hence monotonicity). To guarantee that the refined system is doing nothing more than the original one (up to the new introduced actions and states), any state of it, interpreted as an initial history, should be extended to a state corresponding to an original system state, hence the cofinality requirement.

Refinements of a special form can be defined for trace prefix functions. Consider two concurrent alphabets $(\Sigma_i, \mathrm{Dep}_i)$, where Dep_i is a dependence relation in $\Sigma_i, \mathrm{Ind}_i = \Sigma_i^2 - \mathrm{Dep}_i, i = 1,2$.. Denote by Dep_i^0 the strict dependency in Σ_i, i.e. the relation

$\mathrm{Dep}_i - \Delta_i$, where Δ_i is the identity relation in Σ_i. Trace equivalence class containing a string $w \in \Sigma_i^*$ w.r. to dependency Dep_i is denoted by $[w]_i$, $(i = 1, 2)$.

Let $\psi : \Sigma_1 \longrightarrow \Sigma_2^+$ be a mapping (a substitution), let $\psi^* : \Sigma_1^* \longrightarrow \Sigma_2^*$ be the extension of ψ to strings over Σ_1 defined as usual:

$$\begin{aligned} \psi^*(\epsilon) &= \epsilon, \\ \psi * (wa) &= \psi^*(w)\psi(a) \quad (a \in \Sigma_1, w \in \Sigma_1^*). \end{aligned}$$

For any string w, let $\mathbf{A}(w)$ denotes the set of all symbols occurring in the string w.

Let σ_1 be a trace prefix function over concurrent alphabet $(\Sigma_1, \mathrm{Dep}_1)$, with domain Dom_1 such that $\sigma_1(w) = [w]_1$ for each $w \in \mathrm{Dom}_1$, and $\psi : \Sigma_1 \longrightarrow \Sigma_2^+$ be a substitution mapping meeting the following conditions:

$$\begin{aligned} (a, b) \in \mathrm{Ind}_1 &\Rightarrow \mathbf{A}(\psi(a)) \times \mathbf{A}(\psi(b)) \subseteq \mathrm{Ind}_2, \\ (a, b) \in \mathrm{Dep}_1^0 &\Rightarrow \mathbf{A}(\phi(a)) \times \mathbf{A}(\phi(b)) \subseteq \mathrm{Dep}_2^0. \end{aligned}$$

In other words, substitution ψ respects independence as well as strict dependence of symbols. Such substitutions will be called *concurrency preserving.* Let the domain of trace prefix function σ_2 over concurrent alphabet Σ_2, Dep_2 be

$$\mathrm{Dom}_2 = \mathbf{P}\{w \mid \exists u \in \mathrm{Dom}_1 : w \in [\psi^*(u)]_2\} \qquad (2)$$

and let $\sigma_2(w) = [w]_2$ for each $w \in \mathrm{Dom}_2$. σ_1.

6. *If ψ is a concurrency preserving substitution, domains of trace prefix functions σ_1, σ_2 meet condition (2), then the mapping ϕ : $Rng(\sigma_1) \longrightarrow Rng(\sigma_2)$ such that $\phi([w]_1) = [\psi^*(w)]_2$ is a refinement injection of σ_1 to σ_2.*

Correctness of the definition of ϕ follows from the preservation properties of the substitution ψ that guarantee that

$$[w']_1 = [w'']_1 \Rightarrow [\psi^*(w')]_2 = [\psi^*(w'')]_2,$$

and cofinality of injection follows from the domain definition of σ_2, which ensures that any string $w \in \mathrm{Dom}_2$ is a prefix of a string equivalent to $\psi^*(u)$ for some $u \in \Sigma_1^*$. It means that any trace $[w]_2$ is dominated be the trace $[\psi^*(u)]_2$ for suitable $u \in \Sigma_1^*$.

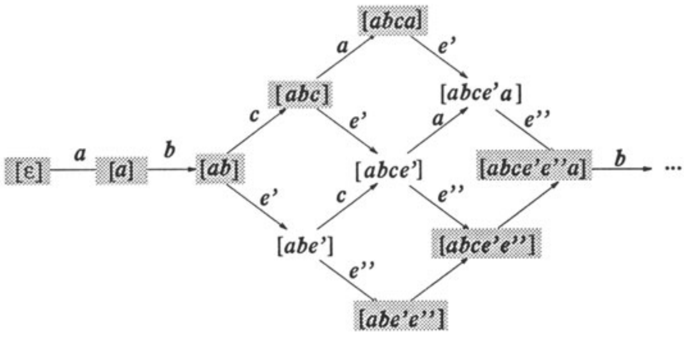

Figure 9 Refinement of a trace prefix function.

Example 6. Consider elementary net system from Example 3, represented graphically in Figure 2. It is described by the trace prefix function σ_1 over alphabet $\{a, b, c, d, e\}$ with dependency $\mathrm{Dep}_1 = \{a, b, c, d\}^2 \cup \{b, e\}^2$, where $\mathrm{Dom}(\sigma_1) = \mathit{Pref}\{w \mid \exists u \in (abce)^* d : w \in [u]_1\}$. Let substitution ψ be such that $\psi(x) = x$ for all $x \in \{a, b, c, d\}$ and $\psi(e) = e'e''$. Then trace prefix function σ_2 over alphabet $\{a, b, c, d, e', e''\}$ with dependency $\mathrm{Dep}_2 = \{a, b, c, d\}^2 \cup \{b, e', e''\}^2$, the domain $\mathrm{Dom}(\sigma_2) = \mathit{Pref}\{w \mid \exists u \in (abce'e'')^* d : w \in [u]_1\}$ is a refinement of σ_1. The diagram of σ_2 is given in Figure 9 and can be compared with that in Figure 3. Injected states are shadowed. □

6 FAIR SEQUENCES AND OBSERVATIONS

The notion of fairness is needed for proving some eventual properties of concurrent systems. To prove that a property of a system states will be eventually reached by the system, one must prove that any system run encounters sooner or later at least a state meeting the required property. Representing runs by sequences of states and properties by sets of states, a property will eventually hold, if any sequence of states representing a system run intersects (has nonempty intersection with) the set of states with the required property. To make this requirement precise, first the notion of a system run should be properly formulated and next, the set of sequences representing runs properly should be defined. The straightforward idea is to represent system runs by arbitrary maximal sequences of states that can come up during the system action. To guarantee maximality of such sequences, each of them should be either infinite or terminating with a state unabling execution of any further action. However, defining such a run as a maximum sequence of the system states (or actions) causes difficulties (Owicki, Lamport 1982). The problem is that not all maximal sequences describe system runs properly. As an example, consider a system consisting of two entirely independent components acting endlessly. It is clear that any infinite sequence containing only states of one component cannot represent the run of the system as a whole, since it ignores action of the second component – we call such a sequence 'unfair'. The problem is how to define 'fair sequences'? A liberal requirement gives rise of too much sequences and hence some obviously inevitable properties will be unable to prove. On the other hand, too restrictive requirements cause acceptability as inevitable some intuitively avoidable properties. Dealing with concurrency expressed explicitely in the system specification, the definition of fair sequences becomes simple and self-explanatory.

Let σ be a (monotonous) prefix function over Σ. A finite or infinite sequence:

$$\mathbf{a} = (a_1, a_2, \ldots, a_n, \ldots), \qquad (a_i \in \Sigma),$$

is said to be *fair* with respect to σ, if the set:

$$V = \{\sigma(\epsilon), \sigma(a_1), \sigma(a_1 a_2), \ldots, \sigma(a_1 a_2 \ldots n), \ldots\}$$

is the *observation* of σ, i.e. if V has the following (completeness) property: if a state of σ is consistent with all elements of V, then it is dominated by some elements of V. Interpreting states as histories, any observation is a set of increasing histories such that any history of a system run is an initial part of some sufficiently large history in the observation. In other words, it means that histories in an observation exhaust the whole history of one (of possibly many) runs of the system. A property of system states is inevitable, if any system observation intersects it. Having defined fair observations, a subset $Q \subseteq \text{Rng}\,(\sigma)$ is *inevitable* in σ, if every observation of σ contains an element of Q. For more details see (Mazurkiewicz, Ochmański, Penczek 1989).

Example 7. Consider prefix function σ_2 from Example 4. Prove that for each n, m the property $\{\sigma_3(w) \mid n \leq w(a), m \leq w(b)\}$ is inevitable w.r. to σ_3. Indeed, since any two states in the range of σ_3 are consistent, any observation for its completeness should contain elements dominating any chosen element of the range. In particular, for completeness such an observation must contain element $\sigma_3(w)$ with $w(a) \geq n, w(b) \geq m$, but it means that any complete observation intersects Q.

Inevitability of Q cannot be proved representing the system by prefix function σ_1 from the same example. Namely, since the only consistent elements of the range of σ_1 are comparable, the sequence with all elements equal to a should be accepted as fair, but the corresponding observation does not intersects Q. Note also that the set $\{\sigma_3(aabb)\}$ is not inevitable w.r. to σ_2; e.g. sequence $(a, a, a, b, b, b, a, b, a, b, \ldots, a, b, \ldots)$ is fair, but the corresponding observation does not intersect the above set. □

7 CONCLUSIONS

Some issues typical for concurrent systems have been presented and discussed. For this purpose a formal model of such systems has been established and used, namely that of prefix function. This model, in spite of its generality, offers a sufficient tool for identification, formalization, and discussion of the some basic concurrency phenomena. Formalization by means of prefix functions can be 'customized': within the same framework different description methods can be used, beginning from interleaving, through traces, multisets, semiwords, to pomsets. Each customized version of prefix functions can serve specific purposes; the choice of suitable version depends upon acual needs. Each version of a prefix function is a filter for looking at concurrent processes from a chosen viewpoint. In such a way prefix function supply us with a unified framework for different approaches to the concurrent systems theory.

The abstraction level of the used model proves the phenomena discussed here to be fundamental for the theory and acceptable as paradigms of concurrent behaviour of systems. No complete results are given in this outline; the aim was only to present some issues and to show a (possible) directions of further investigation that might be helpful in overcoming the difficulties.

8 REFERENCES

Abadi, M., Lamport, L. (1989) Composing Specifications. *Lecture Notes in Computer Science*, **430**,1-41

Diekert, V., Rozenberg, G. (eds.) (1994) *The book of traces.* World Scientific, Singapore,New Jersey,London,Hong Kong.

Mazurkiewicz, A. (1977) Concurrent Program Schemes and Their Interpretation. *Technical Report DAIMI PB-78, Århus University*

Mazurkiewicz, A.(1985) Semantics of Concurrent Systems: A Modular Fixed-Point Trace Approach. *Lecture Notes in Computer Science*, **188**.

Mazurkiewicz, A., Ochmański, E. ,Penczek, W. (1989) Concurrent systems and inevitability.*Theoretical Computer Science*, **64**, 281-304.

Owicki, S., Lamport, L. (1982)Proving liveness properties of concurrent programs. *ACM Trans. Programming, Languages, and Systems*, **4**(3), 455-495.

Petri, C.A. (1977) Non-Sequential Processes.*GMD Report ISF-77-05.*

Pratt, V. (1986) Modeling Concurrency with Partial Orders. *International Journal of Parallel Processing*, **15**, 33-71.

Reisig, W. (1985) *Petri Nets: an Introduction*, EATCS Monographs on Comp.Sci.

Rozenberg, G. (1987) Behaviour of Elementary Net Systems. *Lecture Notes in Computer Science*, **254**, 26-59

Shields, M.W. (1979) *Non-sequential behaviour, part I.* Int. Report CSR-120-82, Dept. of Computer Science, University of Edinburgh.

Thiagarajan, P.S. (1987) Elementary Net Systems. *Lecture Notes in Computer Science*, **254**, 26-59.

Winskel, G. (1988) An introduction to event structures. *Lecture Notes in Computer Science*, **354**, 123-172