SECTION 8

RELATIONSHIP TO OTHER MODELS OF CONCURRENCY

TRACE THEORY

Antoni Mazurkiewicz
Institute of Computer Science
Polish Academy of Sciences
P.O.Box 22, PL-00-901 Warsaw

ABSTRACT. The concept of traces has been introduced for describing non-sequential behaviour of concurrent systems via its sequential observations. Traces represent concurrent processes in the same way as strings represent sequential ones. The theory of traces can be used as a tool for reasoning about nets and it is hoped that applying this theory one can get a calculus of the concurrent processes anologous to that available for sequential systems. The following topics will be discussed: algebraic properties of traces, trace models of some concurrency phenomena, fixed-point calculus for finding the behaviour of nets, modularity, and some applications of the presented theory.

Key words: concurrency, traces, processes, partial ordering, Petri nets.

CONTENTS:

0. INTRODUCTION

In concurrent system activities the ordering of event occurrences is partial: there is no means to decide which one from independent events occurs earlier than the other. The only way to establish objective ordering of event occurrences in such systems is to find their mutual causal dependencies and to agree that a cause must be always earlier than its effect. Therefore, the dependency (or independency) of event occurrences should be a basis for a concurrent behaviour description. On the other hand, a single observer of a concurrent system notices a sequence of events occurring in the system; thus, it would be useful to combine sequential observations (expressed as strings of events) with a dependency relation (expressed as a relation in the set of event occurrences) in order to grasp some concurrency phenomena and to investigate the nonsequential behaviour of systems via their sequential observations.

The algebra of strings and of sets of strings (languages) has been proved to be of great importance for the theory of sequential systems. Many various models of

computations have been investigated by means of sets of strings they can generate (or accept). Such notions as finite automata, sequential machines, regular languages, regular expressions, are strongly connected with strings of symbols representing actions occurring in real systems, since the behaviour of such systems can be described by sets of interpreted strings. The algebra of strings has established a basis for all subsequent theoretical investigations.

In case of concurrent systems, strings of symbols, perfectly suited for modelling sequential processes, are not so generally and directly available. Since some events occurring in concurrent systems are independent of each other, there is no causal relationship putting them in one linearly ordered sequence; instead, their occurrences form a partially ordered set. Therefore an attempt should be made to adjust string oriented methods and techniques to the reality of concurrency.

The theory of traces was motivated by the above arguments. It started in 1977 as a tool for analysing the behaviour of Petri nets. Since then a considerable progress has been made in the theory itself as well as in its applications in related branches of concurrency theory, especially in the theory of Petri nets. On the other hand, trace theory is closely related to the algebraic theory of free partially commutative monoids which is recently intensively developed.

Nowadays, there are several research directions in developing the theory. One is connected with graph representation of traces and the theory of graph grammars; the aim is to overcome some limitations caused by strings and to deal with explicit partial ordering of event occurrences. The second, algebraic, is connected with theory of trace languages, as a basis for concurrent systems specification and description techniques. Issues connected with trace models for temporal logic lead to another research direction.

The paper aims to present briefly some facts and methods offered by trace theory. The proofs of theorems are not included; they are either obvious, or can be found elsewhere. The paper does not pretend to give the whole review of the theory; it would be impossible within the limited size of the paper. The choice of topics reflects the personal interest of the author. The enclosed bibliography gives references to source papers connected with trace theory and those references to papers on Petri nets which are directly related to the topics discussed here.

The standard mathematical denotations are used through the paper. The set of nonnegative integers is denoted by ω. The braces around singletons are omitted.

1. SEQUENTIAL LANGUAGES

1.1. **Strings.** Let A be a finite set of symbols (an alphabet). The ordered triple $\mathbb{S} = (A^*, \circ, \varepsilon)$, where A^* is the set of all finite sequences (strings) of symbols in A, ε is the empty string, and $\circ$ is the concatenation operation on strings, is the well known monoid of strings over A (usually the sign $\circ$ for concatenation is omitted in expressions). Interpreting symbols in alphabet A as elementary (atomic) actions, strings over A can be viewed as composed actions; the effect of such a composed action is the join effect of the constituent atomic actions executed in the sequence indicated by the string.

For each string $w \in A^*$ define

$$Rng(w) = \{a \in A \mid \exists w_1, w_2 \in A^*: w = w_1 a w_2\};$$

Rng(w) is called the range of w. Clearly, for each string w over an alphabet A we have $Rng(w) \subseteq A$ but not necessarily $Rng(w) = A$.

With each string an ordered set can be associated, namely that of symbol occurrences. Let $w = a_1 a_2 \ldots a_n$ be a string, $n \geq 0$; each ordered pair (a_i, n_i) with $n_i = card\{j \mid j \leq i,\ a_i = a_j\}$ is called an occurrence of a_1 in w. The relation

$$Ord(w) = \{\langle (a_i, n_i), (a_j, n_j) \rangle \mid 1 \leq i \leq j \leq n\}$$

is an ordering relation. This ordering will be referred to as the string ordering of w and denoted by Ord(w); it can be interpreted as the ordering of actions execution during a single run of a (sequential) system. Clearly, Ord(w) is a linear ordering for each string w.

Let w be a string over an alphabet A. Elements of the set

$$Pref(w) = \{u \mid \exists v \in A^*: uv = w\}$$

are called prefixes of w. Obviously, $w \in Pref(w)$ and $\varepsilon \in Pref(w)$ for each string w. Let $\sqsubseteq$ be a relation in A^* such that

$$v \sqsubseteq w \iff v \in Pref(w);$$

Clearly, $\sqsubseteq$ is an ordering relation in A^*. This ordering will be called the prefix ordering of strings. For each string w the set Pref(w) is linearly ordered by the prefix ordering. If a string w is interpreted as a composed action execution, elements of Pref(w) can be viewed as all partial executions of that (composed) action; since each such partial execution uniquely determines a state of the system, elements of Pref(w) can be viewed as well as (intermediate) states of the system arising during action w execution.

Let A be an alphabet, $w \in A^*$. For any alphabet B denote by w/B the (string) projection of w onto B, i.e. a string over $A \cap B$ defined as follows:

$$\varepsilon/B = \varepsilon,$$
$$(wa)/B = (w/B)a, \text{ if } w \in A^*, a \in B,$$
$$(wa)/B = (w/B), \text{ if } w \in A^*, a \notin B.$$

Intuitively, projection from A onto B "erases" from a string over A all symbols not belonging to B.

Let u, v be strings; any string

$$u_1v_1u_2v_2\ldots u_nv_n, \quad n \geq 0,$$

such that $u_1u_2\ldots u_n = u$, $v_1v_2\ldots v_n = v$, is called a shuffle of u with v. The set of all shuffles of u with v is denoted by Shuffle(u,v).

1.2. **String languages.** Let A be an alphabet; any subset of A^* is called a (string) language over A. If L_1, L_2 are languages over A, then their concatenation L_1L_2 is defined as

$$L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}.$$

If w is a string, L is a language, then (according to our convention of omitting braces aroud singletons)

$$wL = \{wu \mid u \in L\}, \quad Lw = \{uw \mid u \in L\}.$$

The power of a language L is defined recursively:

$$L^0 = \{\epsilon\},$$
$$L^{n+1} = L^n L,$$

for each $n \in \omega$. The iteration L^* of L is the union:

$$\bigcup_{n\in\omega} L^n.$$

For any language L over A

$$Pref(L) = \bigcup_{w\in L} Pref(w).$$

Elements of $Pref(L)$ are called prefixes of L. Obviously, $L \subseteq Pref(L)$. A language L is prefix closed, if $L = Pref(L)$. Clearly, for any language L the language $Pref(L)$ is prefix closed. An ordered set $(X,\leq)$ is a tree, if:

(i) for each $x \in X$ the set $\{y \in X \mid y \leq x\}$ is linearly ordered;
(ii) for each $x, y \in X$ there is $z \in X$ with $z \leq x$, $z \leq y$.

PROPOSITION 1. For each prefix closed string language L the ordered set $(L,\sqsubseteq)$ is a tree. □

A language L is directed, if for each u, v in L there is w in L such that $u \sqsubseteq w$, $v \sqsubseteq w$.

PROPOSITION 2. Any directed string language is linearly ordered by the prefix relation. □

EXAMPLE 1. The prefix ordering of languages $Pref((ab)^*)$ and $Pref((ac)^*ad)$ are presented in Fig.1 as diagrams (1) and (2); both languages are prefix closed; the first is directed; it represents the infinite sequence (a,b,a,b,a,b,...). Both languages will be discussed later on. □

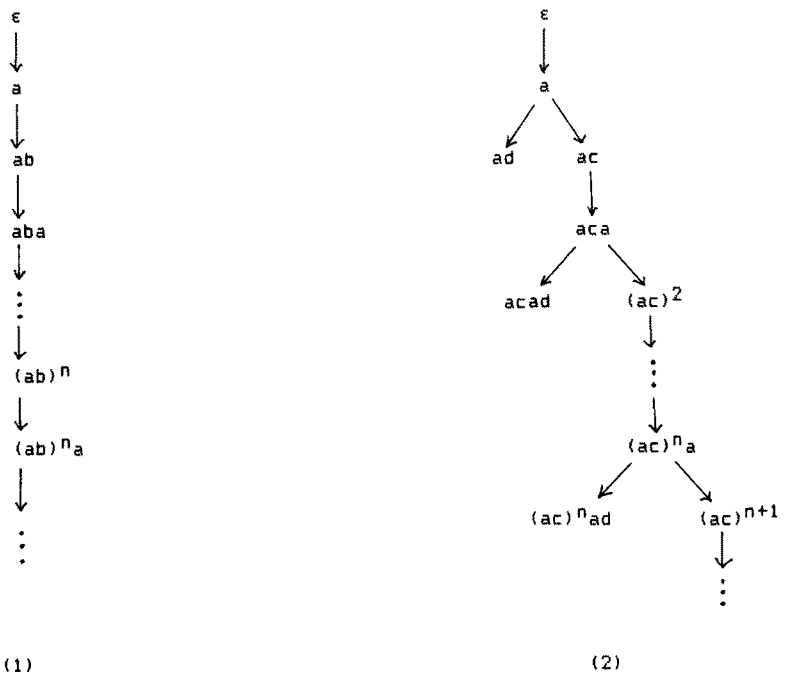
Extend the projection function from strings to languages defining for each language L over an alphabet A and any alphabet B the projection of L onto B as a language L/B over $A \cap B$ defined as

$$L/B = \{u/B \mid u \in L\}.$$

Extend also the notion of shuffling from strings to languages defining for arbitrary

languages L_1, L_2:

$$Shuffle(L_1,L_2) = \bigcup (Shuffle(u,v) \mid u \in L_1,\ v \in L_2).$$



(1) (2)

Fig.1: Prefix structure of Pref((ab)*) (1) and Pref((ac)*(ad)) (2)

1.3. **String systems.** Call a string system, or system for short, any ordered pair (A,L) where A is an alphabet, $L \subseteq A^*$, and $L = \emptyset$ implies $A = \emptyset$. If $X = (A,L)$ is a string system, A is the alphabet of X denoted by $\alpha(X)$ and L is the language of X denoted by $\lambda(X)$. The difference between string languages and string systems is that different systems can have identical languages, as e.g. systems $(\{a\},a^*)$, $(\{a,b\},a^*)$. While a language gives information which sybols are present in strings of the language, a string system gives also information which symbols are absent in those strings; in other words, within systems represented by string systems it is possible to speak not only about executions of some actions but also about their omissions.

Let X be a string system. The prefix ordering of $\lambda(X)$ will be called the prefix

ordering of X and the diagram of this ordering will be called the diagram of X.

Denote by $\emptyset$ and call empty the system $(\emptyset,\emptyset)$. For each system X and each string w over $\alpha(X)$ define

$$wX = (\alpha(X), w\lambda(X)),$$
$$Xw = (\alpha(X), \lambda(X)w);$$

and extend this definition for string languages over $\alpha(X)$:

$$LX = (\alpha(X), L\lambda(X)),$$
$$XL = (\alpha(X), \lambda(X)L)$$

for each string language L over $\alpha(X)$ and each string system X.

We say that a system X is contained in another system Y and write $X \subseteq Y$, if $\alpha(X) \subseteq \alpha(Y)$ and $\lambda(X) \subseteq \lambda(Y)$. Denote the class of all string systems by Ω. Define merging as a binary operation & on string systems:

$$\&: \Omega \times \Omega \longrightarrow \Omega$$

such that for all string systems X, Y

$$\alpha(X\&Y) = \alpha(X) \cup \alpha(Y),$$
$$\lambda(X\&Y) = \{w \in (\alpha(X\&Y))^* \mid w/\alpha(X) \in \lambda(X),\ w/\alpha(Y) \in \lambda(Y)\}$$

(we use here the infix notation and assume & to bind weaker than the concatenation and stronger than other set-theoretical operations). Let X, Y, Z be string systems. The following equalities hold:

$$X\&X = X,$$
$$X\&Y = Y\&X,$$
$$X\&(Y\&Z) = (X\&Y)\&Z$$
$$(\varepsilon,A)\&(\varepsilon,B) = (\varepsilon, A \cup B),$$
$$X\&\emptyset = \emptyset,$$
$$(X \cup Y)\&Z = X\&Z \cup Y\&Z.$$

Observe that for any string systems X, Y

$$\alpha(X) = \alpha(Y) \Rightarrow \lambda(X\&Y) = \lambda(X) \cap \lambda(Y),$$

$$\alpha(X) \cap \alpha(Y) = \emptyset \Rightarrow \lambda(X\&Y) = \mathrm{Shuffle}(\lambda(X),\lambda(Y)).$$

Note also that if $\lambda(X)$, $\lambda(Y)$ are singletons, then in general it is not so in case of $\lambda(X\&Y)$. $\lambda(X\&Y)$ is not empty if and only if

$$\lambda(Y)/\alpha(X) = \lambda(X)/\alpha(Y).$$

EXAMPLE 2. Let X, Y be string systems,

$$X = (\{a,b\},(ab)^*(a \cup \varepsilon)),$$
$$Y = (\{a,c,d\},(ac)^*(ad \cup a \cup \varepsilon));$$

observe that $\lambda(X) = \mathrm{Pref}((ab)^*)$, $\lambda(Y) = \mathrm{Pref}((ac)^*ad)$ (cf. Example 1). The merging of X and Y is the system

$$X\&Y = (\{a,b,c,d\},(abc \cup acb)^*(abd \cup adb \cup ab \cup ac \cup ad \cup a \cup \varepsilon)).$$

The diagram of X&Y is presented in Fig.2 (cf. diagrams from Example 1). □

2. DEPENDENCY AND TRACES

2.1. **Concurrent alphabets.** By a concurrent alphabet we shall mean any ordered pair $\Sigma = (A,D)$ where A is a finite set (the alphabet of Σ) and D is a symmetric and reflexive binary relation in A (the dependency in Σ). In particular, $(\emptyset,\emptyset)$ (the empty alphabet), (A,id_A) and (A,A^2) (alphabets with minimum and full dependency) are concurrent alphabets for each alphabet A. For each concurrent alphabets $\Sigma_1 = (A_1,D_1)$, $\Sigma_2 = (A_2,D_2)$, define

$$\Sigma_1 \cup \Sigma_2 = (A_1 \cup A_2,\ D_1 \cup D_2),$$
$$\Sigma_1 \cap \Sigma_2 = (A_1 \cap A_2,\ D_1 \cap D_2),$$
$$\Sigma_1 \subseteq \Sigma_2 \iff A_1 \subseteq A_2 \text{ and } D_1 \subseteq D_2.$$

It is clear that $\Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2$ are concurrent alphabets, and

$$\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_i \subseteq \Sigma_1 \cup \Sigma_2, \qquad i = 1,2.$$

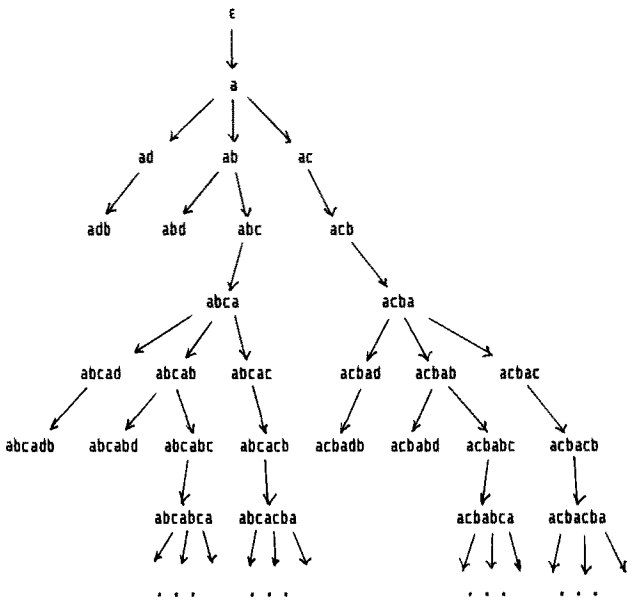


Fig. 2: Prefix structure of Pref((ab)*)&Pref((ac)*ad)

For any alphabet A by D(A) we shall denote the concurrent alphabet (A,A^2) and call it the full dependency alphabet (over A). If $A = \{a_1,a_2,\ldots,a_n\}$, we shall write

$$D(a_1,a_2,\ldots,a_n)$$

instead of D(A). It is clear that each concurrent alphabet is the union of a finite number of full dependency alphabets. Define $I_\Sigma = A^2 \setminus D$; I_Σ will be called the independency in Σ; obviously, I_Σ is a symmetric and irreflexive relation in A. Two symbols a, b in A will be called mutually dependent in Σ, if $(a,b) \in D$; otherwise they are said to be independent in Σ. Any triple (A,I,D) where A is an alphabet, $I \subseteq A^2$ is symmetric and irreflexive, and $D = A^2 \setminus I$ is called a reliance alphabet; this construct is sometimes used as a primitive concept of trace theory and then the concurrent alphabet (A,D) is a derived notion.

EXAMPLE 3. The concurrent alphabet $(\{a,b,c\}, \{a,b\}^2 \cup \{a,c\}^2)$ is the union of D(a,b) and D(a,c); symbols b, c are (the only) independent symbols in this alphabet. □

2.2. **Traces.** Let Σ be a concurrent alphabet. Define $\equiv_\Sigma$ as the least congruence in the monoid $(A^*,\circ,\varepsilon)$ such that

$$(a,b) \in I_\Sigma \Rightarrow ab \equiv_\Sigma ba.$$

In other words, $w' \equiv_\Sigma w''$ if and only if there is a finite sequence of strings

$$(w_0,w_1,\ldots,w_n),$$

$n \geq 0$, such that

$$w_0 = w', \quad w_n = w'',$$

and for each i, $1 \leq i \leq n$, there are strings u, v, and symbols a, b, $(a,b) \in I_\Sigma$, with

$$w_{i-1} = uabv, \; w_i = ubav.$$

The relation $\equiv_\Sigma$ will be referred to as the <u>trace equivalence</u> over Σ.

Equivalence classes of $\equiv_\Sigma$ relation are called <u>traces</u> over Σ; a trace generated by a string w will be denoted by $[w]_\Sigma$ (or by $[w]$, if Σ is understood), and for each $L \subseteq A^*$ define

$$[L]_\Sigma = \{[w]_\Sigma \mid w \in L\}.$$

The set

$$\theta(\Sigma) = [A^*]_\Sigma$$

is the set of all traces over Σ; if Σ is known, we shall write θ instead of $\theta(\Sigma)$.

By definition, a single trace arises by identifying all strings which differ only in the ordering of adjacent independent symbols. If symbols in A are thought as atomic system actions, traces are representing composed actions in which some elementary actions occur independently (i. e. without any causal relationship) of each other. The notion of traces allows to eliminate from strings the ordering between some process actions occurring independently of each other.

EXAMPLE 4. Let $\Sigma = D(a,b) \cup D(a,c)$. Then b, c are independent in Σ; the equivalence

class generated by string abbca is:

$$[abbca]_\Sigma = \{abbca, abcba, acbba\}. \square$$

The quotient algebra $\mathbf{T} = (A^*, \circ, \varepsilon)/\equiv_\Sigma$ is called the algebra of traces over Σ. Clearly, $\mathbf{T}$ is a monoid in which the concatenation operation $\circ$ may be commutative for some different symbols (in contrast to the monoid of strings, where $a \neq b$ implies $ab \neq ba$). The monoid $\mathbf{T}$ is sometimes called a free partially commutative monoid over A. It is clear that in case of full dependency ($D = A^2$) $\mathbf{T}$ is isomorphic to the ordinary algebra $\mathbf{S} = (A^*, \circ, \varepsilon)$ of strings over A. Now, we are going to develop the algebra of traces along the same lines as it was presented above in case of the algebra of strings.

Let Σ be a fixed concurrent alphabet. It is clear that $[w_1] = [w_2]$ implies $Rng(w_1) = Rng(w_2)$; thus, for all strings w we can define

$$Rng([w]) = Rng(w).$$

The following equalities are direct consequences of the definition of traces:

$$[u][w] = [uw],$$
$$\tau[\varepsilon] = [\varepsilon]\tau = \tau,$$
$$\tau_1(\tau_2\tau_3) = (\tau_1\tau_2)\tau_3,$$
$$\sigma'\tau_1\sigma'' = \sigma'\tau_2\sigma'' \iff \tau_1 = \tau_2,$$

where u,w are strings, τ, τ_1, τ_2, τ_3, σ', σ'' are traces. The following equivalence for traces is a generalization of the well-known Levi lemma for strings:

$$\alpha\beta = \gamma\delta \iff \exists \tau_1,\tau_2,\tau_3,\tau_4:$$
$$\alpha = \tau_1\tau_2,\ \beta = \tau_3\tau_4,\ \gamma = \tau_1\tau_3,\ \delta = \tau_2\tau_4,$$
$$Rng(\tau_2) \times Rng(\tau_3) \subseteq I_\Sigma,$$

for any traces α, β, γ, δ. In particular, if $a, b \in A$, we have

$$\tau_1[a] = \tau_2[b],\ a \neq b \iff [ab] = [ba] \text{ and } \exists\sigma:\ \tau_1 = \sigma[b],\ \tau_2 = \sigma[a]$$

for each traces τ_1, τ_2.

2.3. **Dependency graphs.** Dependency graphs are thought as graphical representations of traces, which make explicit the ordering of symbol occurrences within traces. It turns out that, for a given concurrent alphabet, algebra of traces is isomorphic with the algebra of dependency graphs defined below. Therefore, it is only a matter of taste which objects are chosen for representing concurrent processes: equivalence classes of strings or labelled graphs.

Let $\Sigma = (A,D)$ be a concurrent alphabet. Dependency graphs over Σ (or d-graphs for short) are finite, oriented, acyclic graphs with nodes labelled with symbols from A in such a way that two nodes of a graph are connected with an arc if and only if they are different and labelled with dependent symbols. Formally, a triple

$$G = (V,R,\varphi)$$

is a dependency graph, (d-graph) over Σ, if

V is a finite set (of nodes of G),
$R \subseteq V \times V$ (the set of arcs of G),
$\varphi: V \longrightarrow A$ (the labelling of G),

such that

$$R^+ \cap id_V = \emptyset,$$
$$R \cup R^{-1} \cup id_V = \varphi D \varphi^{-1}.$$

Two d-graphs are isomorphic, if there exists a bijection between their nodes preserving labelling and arc connection. As usual, two isomorphic graphs are identified; all properties of d-graphs are given up to isomorphism.

EXAMPLE 5. Let $\Sigma = D(a,b) \cup D(a,c)$. Then the node labelled graph (V,R,φ), with

$$V = \{1,2,3,4,5\}$$
$$R = \{(1,2), (1,3), (1,4), (1,5), (2,4), (2,5), (3,5), (4,5)\},$$
$$\varphi(1) = a,\ \varphi(2) = b,\ \varphi(3) = c,\ \varphi(4) = b,\ \varphi(5) = a,$$

is a d-graph. It is (isomorphic to) the graph in Fig. 3. □

Let $w \in A^*$, Σ be a concurrent alphabet; by $\langle w \rangle_\Sigma$ (or by $\langle w \rangle$, if Σ is understood) denote the node labelled oriented graph defined recursively:

$\langle\varepsilon\rangle$ is the empty graph (no nodes, no arcs);

$\langle we\rangle$ is a graph arising from $\langle w\rangle$ by adding to it a new node labelled with e and new arcs, leading to this new node from all nodes of $\langle w\rangle$ labelled with symbols dependent on e, for each string w and symbol e.

It is clear that the graph $\langle w\rangle$ defined as above is a d-graph for any string w in A^*. Conversely, it turns out that for any d-graph G over Σ there is a string $w \in A^*$ such that G is isomorphic with $\langle w\rangle$.

Therefore,

$$\Gamma_\Sigma = \{\langle w\rangle_\Sigma \mid w \in A^*\}$$

is the class of all d-graphs over Σ.

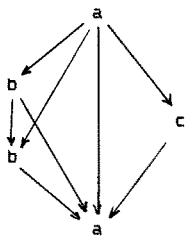


Fig. 3: Dependency graph of [abbca] with bc ≡ cb

The relationship between traces and d-graphs over Σ is expressed by the following theorem.

THEOREM 1. For each strings w', w" in A^*

$$[w']_\Sigma = [w"]_\Sigma \text{ iff } \langle w'\rangle_\Sigma \cong \langle w"\rangle_\Sigma,$$

where $\cong$ denotes the label preserving isomorphism of d-graphs. □

2.4. **Trace ordering.** Let $g = (V,R,\varphi)$ be a d-graph over Σ. Let

$$\xi: V \longrightarrow A \times \omega$$

be a function defined as follows:

$$\xi(v) = (\varphi(v),\#(v)),$$

where

$$\#(v) = \text{card}\,\{u \mid \varphi(u)=\varphi(v),\ (u,v) \in R\} + 1.$$

Thus, ξ assigns to each node of g an occurrence of its label, i.e. an occurrence of a symbol from A. Since each d-graph determines a partial ordering of its nodes, and ξ is a one-to-one function, g determines (via ξ) a partial ordering of symbol occurrences. Denote this ordering by $Ord(g)$; for each string w call $Ord(\langle w\rangle)$ the trace ordering in $[w]$.

THEOREM 2. For each string w: $Ord(\langle w\rangle) = \bigcap_{u\in[w]} Ord(u)$. □

It means that the string orderings in representants of a trace τ are linearizations (extensions to the total ordering) of the trace ordering within τ. In other words, the trace ordering in a trace is the greatest ordering common for all its representants.

Define in Γ_Σ the composition operation $\circ$ as follows: for each graphs g_1, g_2 in Γ_Σ the composition $(g_1 \circ g_2)$ of g_1 with g_2 is a graph arising from the disjoint union of g_1 and g_2 by adding to it new arcs leading from each node of g_1 to each node of g_2 provided they are labelled with dependent symbols.

EXAMPLE 6. Let $\Sigma = D(a,b) \cup D(a,c)$. In Fig. 4 the composition of dependence graphs $\langle abb\rangle$, $\langle ca\rangle$ is depicted; the dotted arrows represent arcs added in the composition. □

THEOREM 3. Given a concurrent alphabet $\Sigma = (A,D)$; the algebra $(\Gamma_\Sigma,\circ,\langle\varepsilon\rangle_\Sigma)$ of dependency graphs over Σ is isomorphic with the algebra $(Tr(\Sigma),\circ,[\varepsilon]_\Sigma)$ of traces over Σ; the isomorphism is established by the correspondence

$$\langle w\rangle_\Sigma \longleftrightarrow [w]_\Sigma$$

for each string $w \in A^*$. □

The dependency graph corresponding to a trace τ will be denoted by $\delta(\tau)$.

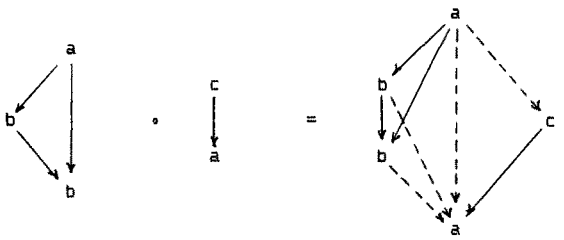


Fig. 4: Composition of dependency graphs

2.5. **Decomposition of traces.** Call two traces τ_1, τ_2 over Σ <u>independent</u>, if

$$Rng(\tau_1) \times Rng(\tau_2) \subseteq I_\Sigma,$$

and <u>dependent</u> otherwise. A trace τ over Σ is <u>connected</u>, if any nonempty traces τ_1, τ_2 with $\tau = \tau_1\tau_2$ are dependent. In other words, τ is connected if $\delta(\tau)$ is a connected graph.

Assign to each trace τ over Σ a set $\Delta_\Sigma(\tau) = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of traces over Σ such that

$\tau = \tau_1\tau_2\ldots\tau_n$,
τ_i is nonempty and connected for each i, $1 \le i \le n$,
τ_i, τ_j are independent for each i, j, $1 \le i < j \le n$.

Elements of $\Delta_\Sigma(\tau)$ will be called <u>independent components</u> of τ, and the set $\Delta_\Sigma(\tau)$ the <u>decomposition</u> of τ. If Σ is known, we shall write $\Delta(\tau)$ rather than $\Delta_\Sigma(\tau)$.

EXAMPLE 7. Let $\Sigma = D(a) \cup D(b)$; then

$$\Delta([abbab]) = \{[aa], [bbb]\}. \square$$

It is clear that in case of full dependency $\Delta(\tau) = \{\tau\}$ for each trace τ.

2.6. **Trace prefixes structure.** Let Σ be a concurrent alphabet. A trace σ is a prefix of a trace τ over Σ, if there is ξ such that $\tau = \sigma\xi$. The set of all prefixes of a trace τ will be denoted by $Pref(\tau)$. The set of all traces over Σ is ordered by a relation $\sqsubseteq \subseteq \theta(\Sigma) \times \theta(\Sigma)$ defined as

$$\tau_1 \sqsubseteq \tau_2 \iff \tau_1 \in Pref(\tau_2).$$

Relation $\sqsubseteq$ will be referred to as the dominating relation in $\theta(\Sigma)$ and if $\tau_1 \sqsubseteq \tau_2$ we say that τ_1 is dominated by τ_2 or that τ_2 dominates τ_1. In contrast to linearly ordered prefixes of a single string the set $Pref(\tau)$ is ordered by $\sqsubseteq$ only partially.

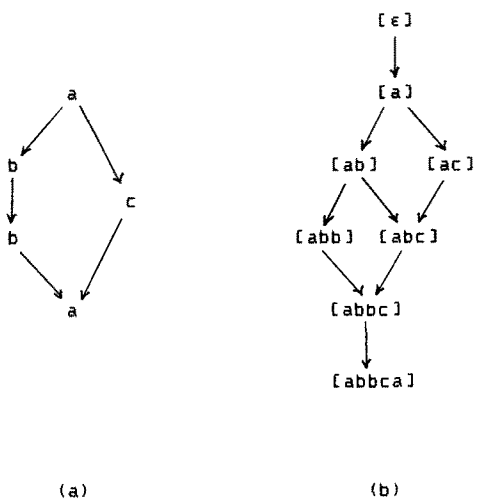


Fig. 5: Dependency graph (a) and prefix structure (b) of [abbca] with bc ≡ cb

EXAMPLE 8. Let $\Sigma = D(a,b) \cup D(a,c)$. In Fig. 5 the dependency graph of trace [abbca] and the prefix structure of [abbca] are presented. In both diagrams the arrows following from others by transitivity or reflexivity are omitted. □

While the ordering given by the dependency graph of a trace can be interpreted as the ordering of action occurrences in a process, that given by the prefix ordering of the same trace can be viewed as the (global) state ordering in this process; $[\varepsilon]$ represents the initial state; incomparable prefixes represent states arising in effect of the concurrent actions execution; if τ_2 dominates τ_1, τ_1 is the effect of a partial execution of a process leading to τ_2.

2.7. **Trace projections.** Let Σ_1, Σ_2 be concurrent alphabets, $\Sigma = \Sigma_1 \cap \Sigma_2$, A be the alphabet of Σ, τ be a trace over Σ_1, $\tau = [w]$. Denote by τ/Σ_2 the (trace) projection of τ onto Σ_2, i.e. a trace over Σ, defined by the following equality

$$[w]_{\Sigma_1}/\Sigma_2 = [w/A]_{\Sigma}.$$

It can be easily checked that the above definition is correct, i.e. that if $[w]_{\Sigma_1} = [u]_{\Sigma_1}$, then $[w]_{\Sigma_1}/\Sigma_2 = [u]_{\Sigma_1}/\Sigma_2$.

Let Σ_1, Σ_2 be concurrent alphabets, $\Sigma = \Sigma_1 \cup \Sigma_2$. The following equalities hold:

$$\tau/(\emptyset,\emptyset) = [\varepsilon]_{(\emptyset,\emptyset)},$$
$$(\tau\sigma)/\Sigma = (\tau/\Sigma)(\sigma/\Sigma),$$
$$((\tau/\Sigma_1)/\Sigma_2 = \tau/(\Sigma_1 \cap \Sigma_2).$$

for all traces τ, σ over Σ.

3. TRACE SYSTEMS

3.1. **Trace languages.** Let Σ be a concurrent alphabet, A be the alphabet of Σ. Each subset of $\theta(\Sigma)$ is called a trace language over Σ. Concatenation of trace languages is defined in usual way; for each trace languages T, S, their concatenation is the language

$$TS = \{\tau_1\tau_2 \mid \tau_1 \in T,\ \tau_2 \in S\}.$$

For each nonnegative integer n the power T^n of a trace language T over Σ is defined recursively:

$$T^0 = [\varepsilon]_\Sigma,$$
$$T^{n+1} = T^n T, \quad n \in \omega.$$

The iteration of T is the union

$$T^* = \bigcup_{n\in\omega} T^n.$$

The following formulas are valid for arbitrary (string) languages over A: L, L_1, L_2, and an arbitrary family $(L_i)_{i\in J}$ of such languages:

$$[\emptyset] = \emptyset,$$
$$[L_1][L_2] = [L_1L_2],$$
$$L_1 \subseteq L_2 \Rightarrow [L_1] \subseteq [L_2],$$
$$[L_1] \cup [L_2] = [L_1 \cup L_2],$$
$$\bigcup_{i\in J} [L_i] = [\bigcup_{i\in J} L_i],$$
$$[L]^* = [L^*].$$

For each trace language T define

$$Pref(T) = \bigcup_{\tau\in T} Pref(\tau).$$

A trace language T is <u>prefix closed</u>, if T = Pref(T); in other words, T is prefix closed, if

$$\tau \in T, \sigma \sqsubseteq \tau \Rightarrow \sigma \in T.$$

A trace language T is <u>directed</u>, if

$$\tau_1 \in T, \tau_2 \in T \Rightarrow \exists \tau\in T: \tau_1 \sqsubseteq \tau, \tau_2 \sqsubseteq \tau$$

3.2. **Generalized dependency graphs.** Similarly as infinite strings can be defined as directed and prefix closed string languages (consisting of all finite prefixes of such an infinite string), infinite traces can be defined as directed and prefix closed trace languages; an infinite trace can be identified with a trace language consisting of all finite trace prefixes of such a trace. The notion of dependency graph can be easily generalized to this case. To do it, we need some auxiliary definitions. Let Σ be a concurrent alphabet; we shall consider oriented and acyclic graphs with nodes labelled with symbols from Σ. A subgraph G_0 of such a graph G is the <u>initial part</u> of

G, if, together with a node v, G_0 contains all arcs in G leading to v. The class of all initial parts of a graph G will be denoted by Init(G).

THEOREM 4. For each directed and prefix closed trace language T over Σ there exists unique (up to isomorphism) oriented and acyclic graph G with nodes labelled with symbols in Σ such that for each γ

$$\exists \tau \in T\colon \delta(\tau) \cong \gamma \iff \exists g \in Init(G)\colon g \cong \gamma$$

for each γ ($\cong$ denotes the isomorphism of oriented labelled graphs). □

For each directed trace language T over Σ denote by $\delta(T)$ and call the d-graph of T the graph the existence of which is ensured by Theorem 4. Clearly, $\delta(T)$ need not be finite. Dependency graphs of directed trace languages are intended to describe explicitly the ordering of symbols (elementary actions) in processes represented by such languages.

EXAMPLE 9. Let $\Sigma = D(a,b) \cup D(a,c)$. The trace language $T = [(abc)^*]$ over Σ is directed; $\delta(T)$ is the (infinite) graph given in Fig. 6 (as usual, arcs following by transitivity from others are omitted). □

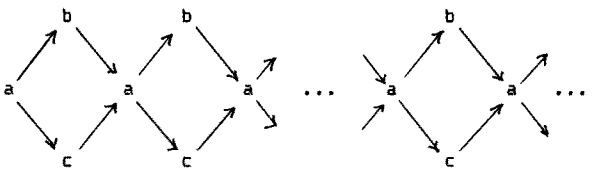


Fig. 6: An infinite dependency graph

3.3. **Regular trace languages.** Let $T \subseteq \theta(\Sigma)$; the decomposition of T is defined as the set

$$\Delta_\Sigma(T) = \bigcup_{\tau \in T} \Delta_\Sigma(\tau).$$

We write $\Delta(T)$ instead of $\Delta_\Sigma(T)$, if Σ is understood. From the above definition it follows that $\Delta(T)$ contains connected traces only. Furthermore, operation Δ has the

following properties:

$T^* \subseteq \Delta(T)^*$;
$T = \Delta(T)$, if the dependency in Σ is full;
$\Delta(\Delta(T)) = \Delta(T)$.

We say that a trace language T is connected, if $T = \Delta(T)$. Obviously, if T, S are connected trace languages, then $T \cup S$ is also connected.

EXAMPLE 10. Let $\Sigma = D(a) \cup D(b)$. Then the language $[ab]$ is not connected, since $[a] \in \Delta([ab]) = \{[a], [b]\}$ and $[a] \notin [ab]$; the language $[a \cup b]$ is connected. □

The trace iteration $T^\#$ of a trace language T is defined as

$$T^\# = (\Delta(T))^*.$$

Roughly speaking, in $T^\#$ connected components of a trace in T are iterated independently of each other (being treated as separate elements of T), in contrast to the ordinary iteration T^*, where all components of a trace in T are iterated simultaneously. If dependency is full, then clearly $T^* = T^\#$.

EXAMPLE 11. Let $\Sigma = D(a) \cup D(b)$. Then $[ab]^* \neq [ab]^\#$, since $[ab]^* = \{[a^n b^n] \mid n \in \omega\}$, while $[ab]^\# = \{[a^n b^m] \mid n \in \omega, m \in \omega\} = [a^* b^*]$. □

Let T be a trace language over a concurrent alphabet Σ. For each trace τ over Σ denote by (T/τ) the set

$$\{\sigma \in \theta(\Sigma) \mid \tau\sigma \in T\}.$$

Similarly to the case of string languages, call a trace language T regular, if the family of sets

$$\{(T/\tau) \mid \tau \in \theta(\Sigma)\}$$

is finite. The following theorem characterizes regular trace languages in a similar way to the characterization of regular string languages; in fact, the following theorem is a generalization of the Kleene characterization of regular string languages.

THEOREM 5 (Ochmański 1985). The class of all regular trace languages over Σ is the

least class of languages containing all finite trace languages over Σ and closed w.r. to the union, concatenation, and the trace iteration operations. □

Observe that in case of full dependency, i.e. when traces are isomorphic with strings, this theorem is reduced to the Kleene theorem, since in this case trace and string iterations are identical (up to isomorphism).

3.4. **Fixed-point equations.** Let $\Sigma_1 = (A,D_1)$, $\Sigma_2 = (B,D_2)$ be concurrent alphabets and let φ be a function:

$$\varphi\colon 2^{A^*} \longrightarrow 2^{B^*}.$$

We say that φ is <u>monotonic</u>, if for each $X, Y \subseteq A^*$

$$X \subseteq Y \Rightarrow \varphi(X) \subseteq \varphi(Y),$$

and that φ is <u>congruent</u> (w.r. to Σ_1, Σ_2), if for each $X, Y \subseteq A^*$

$$[X]_{\Sigma_1} = [Y]_{\Sigma_1} \Rightarrow [\varphi(X)]_{\Sigma_2} = [\varphi(Y)]_{\Sigma_2}.$$

If φ is congruent, then there exists a function ψ,

$$\psi\colon 2^{\theta(\Sigma_1)} \longrightarrow 2^{\theta(\Sigma_2)}$$

such that for each $X \subseteq A^*$

$$\psi([X]_{\Sigma_1}) = [\varphi(X)]_{\Sigma_2}.$$

Clearly, arbitrary superpositions of monotonic and congruent functions are monotonic and congruent. For any set X and any function

$$\varphi\colon 2^X \longrightarrow 2^X,$$

a subset X_0 of X is the least fixed point of φ, if

$$\varphi(X_0) = X_0,$$

and for each $Y \subseteq X$

$$\varphi(Y) = Y \Rightarrow X_0 \subseteq Y.$$

The following theorem expresses the relationship between solving equations (finding fixed points) in the domain of string languages and that of trace languages.

THEOREM 6. Let $\Sigma = (A,D)$ be a concurrent alphabet. If

$$\varphi: 2^{A^*} \longrightarrow 2^{A^*}$$

is monotonic and congruent, X_0 is the least fixed point of φ, then $[X_0]_\Sigma$ is the least fixed point of function $[\varphi]_\Sigma$ defined for all $X \subseteq A^*$ by the equality

$$[\varphi]_\Sigma([X]_\Sigma) = [\varphi(X)]_\Sigma. \quad \square$$

This theorem can be easily generalized for tuples of monotonic and congruent functions. As examples of monotonic and congruent functions can serve functions built up from variables and constants by means of union, concatenation and iteration operations. Another example of congruent functions are trace projections.

3.5. **Trace systems.** By a trace system we shall understand any ordered pair (Σ,T), where Σ is a concurrent alphabet and $T \subseteq \theta(\Sigma)$. The class of all trace systems will be denoted by $\mathcal{Z}$. Similarly to string systems, let for each trace system $X = (\Sigma,T)$

$$\alpha(X) = \Sigma, \ \lambda(X) = T.$$

We shall write $\emptyset$ for the empty trace system $(\emptyset,(\emptyset,\emptyset))$ and $X \subseteq Y$, if $\alpha(X) \subseteq \alpha(Y)$ and $\lambda(X) \subseteq \lambda(Y)$. Clearly, $\mathcal{Z}$ is partially ordered by $\subseteq$. A trace system X is said to be: sequential, if $\alpha(X) = D(A)$ for some alphabet A, i.e. if the dependency in $\alpha(X)$ is full; regular (directed, prefix closed), if $\lambda(X)$ is a regular (directed, prefix closed, resp.) trace language over $\alpha(X)$. If τ is a trace in $\theta(\alpha(X))$, define

$$\tau X = (\alpha(X),\tau\lambda(X)), \ X\tau = (\alpha(X),\lambda(X)\tau);$$

and extend this definition for trace languages:

$$TX = (\alpha(X),T\lambda(X)), \ XT = (\alpha(X),\lambda(X)T),$$

for each trace language T over $\alpha(X)$.
The motivation for introducing trace systems is the same as that for string systems:

within trace systems we can speak not only about occurrences of sybols in traces, but also about their absence.

3.6. **Synchronization.** The synchronization operation is a trace analogue of the merging operation for string systems. By synchronization we mean a binary operation in $\mathcal{Z}$, i.e. a function

$$\|: \Pi \times \Pi \longrightarrow \Pi$$

such that for each trace systems X, Y

$$\alpha(X\|Y) = \alpha(X) \cup \alpha(Y),$$
$$\lambda(X\|Y) = \{\tau \in \theta(\alpha(X\|Y)) \mid (\tau/\alpha(X)) \in \lambda(X), (\tau/\alpha(Y)) \in \lambda(Y)\}.$$

The infix notation is used here for $\|$ operation; $\|$ is assumed to bind weaker than the concatenation and stronger than other set-theoretical operations. Let X, Y, Z be trace systems, Σ_1, Σ_2 be concurrent alphabets, τ be a trace over $\alpha(X) \cup \alpha(Y)$. The following equalities hold:

$$X\|X = X,$$
$$X\|Y = Y\|X,$$
$$X\|(Y\|Z) = (X\|Y)\|Z$$
$$([\varepsilon],\Sigma_1)\|([\varepsilon],\Sigma_2) = ([\varepsilon],\Sigma_1 \cup \Sigma_2),$$
$$X\|\emptyset = \emptyset,$$
$$(X \cup Y)\|Z = X\|Z \cup X\|Z,$$
$$\tau(X\|Y) = (\tau/\alpha(X))X\|(\tau/\alpha(Y))Y),$$
$$(X\|Y)\tau = X(\tau/\alpha(X))\|Y(\tau/\alpha(Y)).$$

The essential difference between the (string) merging & and the (trace) synchronization $\|$ is that the last operation applied to singletons returns either a singleton or the empty set, which (in general) is not true for the merging operation; it means that $\|$ restricted to singletons is a partial function, while & similarly restricted is not.

The following propositions express some preservation properties of the synchronization operation.

PROPOSITION 4. If X, Y are regular trace systems, then so is $X\|Y$. □

PROPOSITION 5. If X, Y are prefix closed trace systems, then so is X‖Y. □

The next theorem enables us to find the synchronization of trace languages by solving fixed point equations.

THEOREM 7. The synchronization operation ‖ is the least binary operation in Z (w.r. to the inclusion ordering of Z) meeting the following properties:

(i) $([\varepsilon],A)\|([\varepsilon],B) = ([\varepsilon],A \cup B)$

(ii) $(X \cup Y)\|Z = X\|Z \cup Y\|Z$,

(iii) $\tau(X\|Y) = (\tau/\alpha(X))X\|(\tau/\alpha(Y))Y$,

(iv) $X\|Y = Y\|X$,

for any concurrent alphabets A, B, any trace systems X, Y, Z, and each trace τ in $\theta(\alpha(X) \cup \alpha(Y))$. □

EXAMPLE 12. Let A = D(a,b), B = D(a,c,d) and let X = $(A,[(ab)^*(a \cup \varepsilon)])$, Y = $(B,[(ac)^*(ad \cup a \cup \varepsilon)])$ be trace systems. Find X‖Y.

First, put $X_0 = (A,[\varepsilon])$, $Y_0 = (B,[\varepsilon])$, and $Z_0 = (A \cup B,[\varepsilon])$. Next, write equations for X and Y:

$$X = [ab]X \cup ([a] \cup [\varepsilon])X_0,$$
$$Y = [ac]Y \cup ([ad] \cup [a] \cup [\varepsilon])Y_0.$$

Hence,

$$X\|Y = ([ab]X \cup [a]X_0 \cup X_0)\|([ac]Y \cup [ad]Y_0 \cup [a]Y_0 \cup Y_0);$$

by (ii) we have

$$\begin{aligned} X\|Y = {} & ([ab]X\|[ac]Y) \cup ([a]X_0\|[ac]Y) \cup (X_0\|[ac]Y) \cup \\ & ([ab]X\|[a]Y_0) \cup ([a]X_0\|[a]Y_0) \cup (X_0\|[a]Y_0) \cup \\ & ([ab]X\|[ad]Y_0) \cup ([a]X_0\|[ad]Y_0) \cup (X_0\|[ad]Y_0) \cup \\ & ([ab]X\|Y_0) \cup ([a]X_0\|Y_0) \cup (X_0\|Y_0); \end{aligned}$$

by (i) and (iii) we get

$$[ab]X\|[ac]Y = [abc](X\|Y);$$

$$[a]X_0 \| [ac]Y = [ac](X_0 \| Y);$$
$$X_0 \| [ac]Y = \emptyset;$$
$$[ab]X \| [a]Y_0 = [ab](X \| Y_0);$$
$$[a]X_0 \| [a]Y_0 = [a](X_0 \| Y_0);$$
$$X_0 \| [a]Y_0 = \emptyset;$$
$$[ab]X \| [ad]Y_0 = [abd](X \| Y_0);$$
$$[a]X_0 \| [ad]Y_0 = [ad](X_0 \| Y_0);$$
$$X_0 \| [ad]Y_0 = \emptyset;$$
$$[a]X_0 \| Y_0 = \emptyset.$$

Thus, we have

$$X \| Y = [abc](X \| Y) \cup [ac](X_0 \| Y) \cup [ab](X \| Y_0) \cup [a](X_0 \| Y_0) \cup [abd](X \| Y_0) \cup [ad](X_0 \| Y_0).$$

Now we need to calculate $X_0 \| Y_0$, $X_0 \| Y$, and $X \| Y_0$:

$$X_0 \| Y_0 = Z_0, \text{ (by (i))},$$
$$X_0 \| Y = X_0 \| ([ac]Y \cup [a]Y_0 \cup [ad]Y_0 \cup Y_0) = X_0 \| Y_0 = Z_0,$$
$$X \| Y_0 = ([ab]X \cup [a]X_0 \cup X_0) \| Y_0 = X_0 \| Y_0 = Z_0.$$

Denote $X \| Y$ by Z; then Z satisfies the following equation:

$$Z = [abc]Z \cup ([ab] \cup [ac] \cup [abd] \cup [ad] \cup [a] \cup [\epsilon])Z_0.$$

By Theorem 6 and Theorem 7 the least solution of the above equation is

$$\begin{aligned}
&[(abc)^*(abd \cup ab \cup ad \cup ac \cup a \cup \epsilon]Z_0 \\
&\quad = Pref[(abc)^*abd]Z_0 \\
&\quad = (A \cup B, Pref[(abc)^*abd]) \\
&\quad = (A \cup B, Pref([abc]^*[abd])).
\end{aligned}$$

Since $[abc]$ is a connected trace language over $A \cup B$ we have as well

$$X \| Y = (A \cup B, Pref([abc]^{\#}[abd]))$$

which proves the regularity of $X \| Y$. Observe that X and Y are sequential trace systems, but $X \| Y$ is not. Since $\alpha(X \| Y) = A \cup B = D(a,b) \cup D(a,c,d)$, (b,c) and (b,d) are independent in this system. The diagram in Fig. 6 represents the prefix ordering of $X \| Y$. □

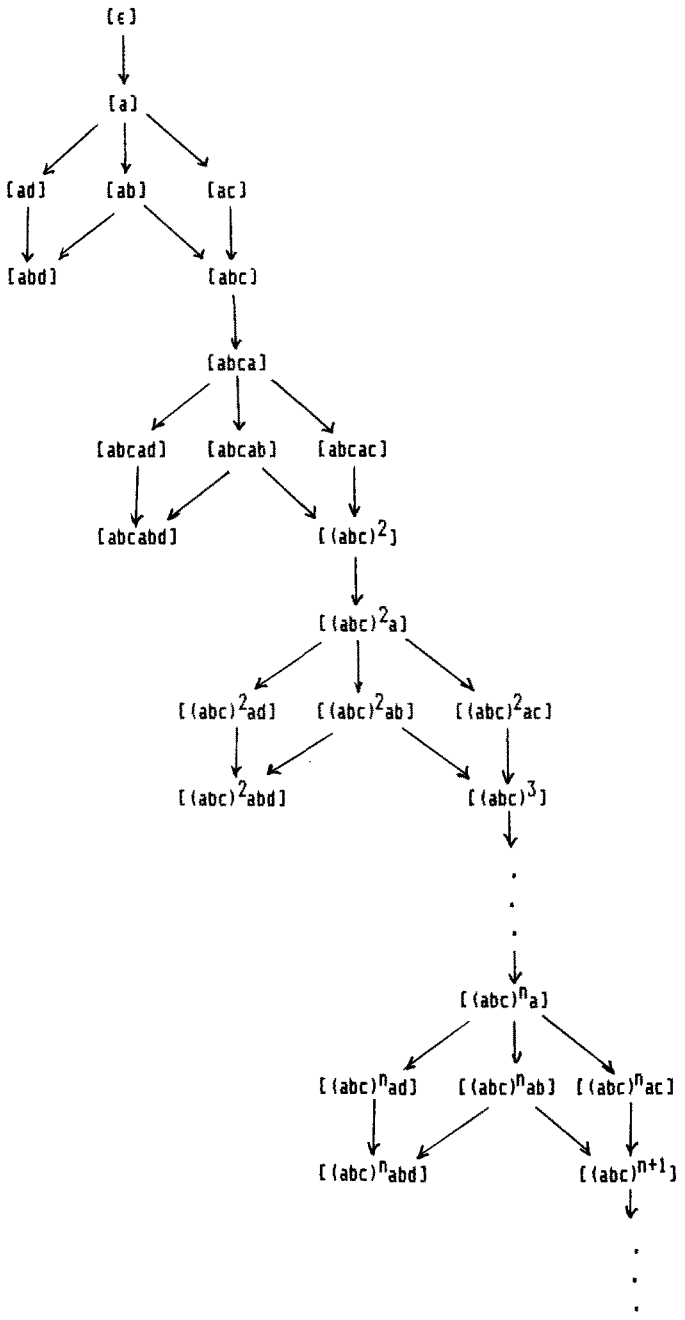


Fig. 6: Prefix structure of Pref[(abc)*abd] with bc ≡ cb, bd ≡ db.

4. NETS AND TRACES

4.1. Elementary net systems. By an elementary net system (EN system, for short), we understand any ordered quadruple

$$N = (B,E;F,C_{in}),$$

where B and E are finite, disjoint, nonempty sets (of conditions and events of N, respectively), $F \subseteq B\times E \cup E\times B$ (the flow relation of N), with $dom(F) \cup cod(F) = B \cup E$, and $C_{in} \subseteq B$ (the initial case of N). As all other types of nets, EN systems are represented graphically using boxes to represent events, circles to represent conditions, and arrows leading from circles to boxes or from boxes to circles to represent the flow relation; in such a representation circles corresponding to conditions in the initial case are marked with dots.

EXAMPLE 13. The following quadruple is an EN system:

$$N = (\{1,2,3,4\}, \{a,b,c,d\};$$
$$\{(1,a),(a,3),(3,b),(b,1),(2,a),(a,4),(4,c),(4,d),(c,2)\}, \{1,2\}).$$

In Fig. 7 this system is represented graphically. □

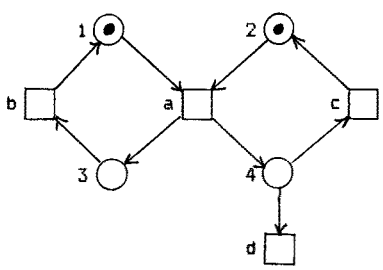


Fig. 7: An elementary net system

Let $N = (B,E;F,C_{in})$ be an EN system. For each $x \in B \cup E$, the sets:

$$\begin{aligned} {}^{\bullet}x &= \{y \mid (y,x) \in F\}, \\ x^{\bullet} &= \{y \mid (x,y) \in F\}, \\ {}^{\bullet}x^{\bullet} &= {}^{\bullet}x \cup x^{\bullet}, \end{aligned}$$

are called respectively the preset, the postset, and the proximity of x. Two EN systems are said to be B-disjoint (E-disjoint), if their sets of conditions (events, resp.) are disjoint.

4.2. **Firing sequences.** Let $N = (B,E;F,C_{in})$ be an EN system. The transition function δ_N of N is a partial function:

$$\delta_N: 2^B \times E \longrightarrow 2^B$$

such that

$$\delta_N(C,e) = (C \setminus {}^{\bullet}e) \cup e^{\bullet}, \text{ if } {}^{\bullet}e \subseteq C,\ e^{\bullet} \cap C = \emptyset, \text{ and undefined otherwise.}$$

Function δ_N describes a change of conditions holding caused by a single event occurrence, if it is defined; otherwise the event has no concession to occur. The reachability function R_N of N is a partial function:

$$R_N: E^* \longrightarrow 2^B,$$

such that

$$\begin{aligned} R_N(\varepsilon) &= C_{in}, \\ R_N(we) &= \delta_N(R_N(w),e), \end{aligned}$$

for all $e \in E$, $w \in E^*$. Set

$$S_N = Dom(R_N),\ C_N = Cod(R_N),$$

and call elements of S_N (i.e. some sequences of events) firing sequences of N, and those of C_N (i.e. some sets of conditions) cases of N. If w is a firing sequence and $R_N(w) = C$, then w is said to lead to the case C of N.

The string system (E,S_N) will be referred to as the sequential behaviour of N and denoted by SB_N. Clearly, SB_N is prefix closed and ordered into a tree by the prefix

ordering. Maximal directed subsets of S_N are then full paths through this tree and can be viewed as sequential observations of the behaviour of N, i.e. observations made by observers capable to see only a single event occurrence at a time. The ordering of symbols in strings of SB_N reflects not only the (objective) causal ordering of event occurrences but also a (subjective) observational ordering resulting from a specific view over concurrent actions. Therefore, the structure of SB_N alone does not allow to decide whether the difference in ordering is caused by a conflict resolution (a decision made in the system), or by different observations of concurrency. In order to extract from SB_N the causal ordering of event occurrences we must supply SB_N with an additional information; as such information we take here the dependency of events.

4.3. **Firing traces.** Let $N = (B,E;F,C_{in})$ be an EN system fixed for this section. Define dependency in N as a relation $D_N \subseteq E \times E$ such that:

$$(e_1,e_2) \in D_N \iff {}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset.$$

Clearly, D_N is a reflexive and symmetric. The complement of D_N in E will be called the independency in N. By definition, two events are independent in N, if and only if they have no adjacent conditions in common; if such events occur next to each other in a firing sequence, the order of their occurrences is irrelevant, since (according to our understanding of EN system activity) they occur concurrently in this execution. In order to get rid of this ordering we intend to replace the sequential description of EN systems behaviour by nonsequential one, expressing the behaviour of EN systems by traces rather than sequences. To this end, set $\Sigma_N = (E,D_N)$ and call Σ_N the concurrent alphabet associated with N; let $\equiv_N$ denote the trace equivalence over Σ_N and $[w]_N$ the equivalence class of $\equiv_N$ represented by w. As usual, the subscript N will be omitted if it causes no ambiguity.

The trace behaviour of N is defined as a trace system $TB_N = (\Sigma_N,[S_N])$; elements of TB_N will be called firing traces of N.

It is clear that each firing sequence of N is contained in some equivalent class belonging to TB_N. Conversely, from the theorem below it follows that each representant of a firing trace of N is a firing sequence of N and all equivalent firing sequences lead to the same case of N.

THEOREM 8. The reachability function R_N is congruent w.r. to Σ_N, i.e. for each two strings w_1, w_2 in E*

$$w_1 \equiv_N w_2 \Rightarrow R_N(w_1) = R_N(w_2)$$

(here, the equality in the conclusion means that either both sides are defined and equal, or both of them are undefined). □

This is another motivation for introducing traces for describing concurrent systems behaviour. Representing traces by their dependency graphs, firing traces can be viewed as partially ordered sets of symbol occurrences; the ordering within firing traces is determined by mutual dependencies of events and then it reflects the causal relationship between event occurrences rather than the non - objective ordering following from the string representation of the EN system activity.

PROPOSITION 6. TB_N is a prefix closed trace system. □

Let Σ be a concurrent alphabet. Two traces over Σ are said to be consistent, if both of them are prefixes of a common trace over Σ. A trace system (Σ,X) is proper, if each two consistent traces in X are prefixes of a common trace in X. Levi Lemma for traces and the transition function definition implies the following proposition.

PROPOSITION 7. TB_N is a proper trace system. □

Each firing trace uniquely determines a case of the EN system reached from its initial case. Moreover, each firing trace determines uniquely the way of reaching this case (the history of the EN system activity resulting with this case). Therefore, firing traces can be identified with (global) states of an EN system. In this approach a state defines, roughly speaking, what has already happened in the EN system. This identification would be impossible while dealing with firing sequences, since some different strings might represent the same history of the considered system; the one to one correspondence between firing sequences and states would not be achieved before identifying some of sequences into equivalence classes.

Directed subsystems of TB_N will be called (trace) processes in N. If P is a process in N, then by its directedness and prefix closedness there exists a (generalized) dependency graph of P; thus, processes in N can be viewed as (possibly infinite) dependency graphs over Σ_N. The set of all processes in N describes all possible executions of the system represented by N.

4.4. Composition and decomposition of EN systems. In this section a modular way of finding the behaviour of EN systems is presented. It will be shown how to construct

the behaviour of an EN system from behaviours of some parts of this EN system. The method is to decompose a given EN system into modules, to find their behaviours, to put them together, and to get in this way the behaviour of the original EN system.

We say that an EN system $N = (E,B;F,C_{in})$ is composed of EN systems $N_1 = (E_1,B_1;F_1,(C_{in})_1)$, $N_2 = (E_2,B_2;F_2,(C_{in})_2)$, and write

$$N = N_1 + N_2,$$

if N_1, N_2 are B-disjoint (i.e. $B_1 \cap B_2 = \emptyset$), $E = E_1 \cup E_2$, $B = B_1 \cup B_2$, $F = F_1 \cup F_2$, and $C_{in} = (C_{in})_1 \cup (C_{in})_2$. Obviously, $(C_{in})_1 \cap (C_{in})_2 = \emptyset$. If $N = N_1 + N_2$, N_1 and N_2 are called components of N.

PROPOSITION 8. For each pairwise B-disjoint EN systems N_1, N_2, N_3:

$$N_1 + N_2 = N_2 + N_1,$$
$$(N_1 + N_2) + N_3 = N_1 + (N_2 + N_3). \quad \square$$

Thus, + is a symmetric and associative operation on B-disjoint EN systems; it makes possible to omit parentheses and disregard the order of terms in composition.

PROPOSITION 9. For each B-disjoint EN systems N_1, N_2:

$$\Sigma_{N_1+N_2} = \Sigma_{N_1} \cup \Sigma_{N_2}. \quad \square$$

The main theorem of this section is the following:

THEOREM 9. If N_1, N_2, ..., N_k are pairwise B-disjoint EN systems, then

$$TB_{N_1+N_2+\ldots+N_k} = TB_{N_1} \| TB_{N_2} \| \ldots \| TB_{N_k}. \quad \square$$

This theorem allows us to find the behaviour of an EN system knowing behaviours of its components. One can expect the behaviour of components to be easier to compute than that of the whole EN system. In fact, there exists a standard set of very simple EN systems, with behaviours already known, such that an arbitrary EN system can be composed from systems in this set; these EN systems, containing only one condition, will be called atomic EN systems or simply atoms.

Let $N = (E,B;F,C_{in})$ be an EN system; for each $b \in B$ the EN system

$$N_b = ({}^\bullet b^\bullet, \{b\}, F_b, (C_{in})_b),$$

where

$$F_b = \{(e,b) \mid e \in {}^\bullet b\} \cup \{(b,e) \mid e \in b^\bullet\},$$
$$(C_{in})_b = C_{in} \cap \{b\},$$

is called an atom of N (determined by b). Clearly, the following proposition holds:

PROPOSITION 10. Each EN system is the composition of all its atoms. □

In Fig. 8 atoms of the EN system from Example 13 are presented.

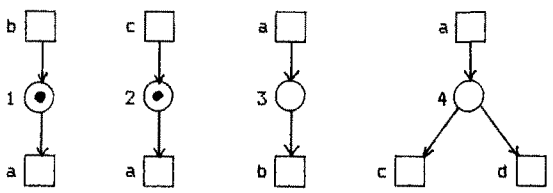


Fig. 8: Atomic EN systems

PROPOSITION 11. Let $N = (B,E;F,C_{in})$ be an EN system, $b \in B$, $U = {}^\bullet b$, $V = b^\bullet$, $A = {}^\bullet b^\bullet$. Then

$$TB_{N_b} = (D(A), [(VU)^*(V \cup \epsilon)]), \quad \text{if } b \in C_{in} \text{ and}$$
$$TB_{N_b} = (D(A), [(UV)^*(U \cup \epsilon)]), \quad \text{if } b \notin C_{in}. \ \square$$

This proposition together with Theorem 9 is to the effect that the activity of an arbitrary EN system can be found by performing the synchronization operation applied to trace behaviours of atoms of the system; these behaviours are given by the above expressions. Thus, for an arbitrary EN system it is possible to write down an explicit expression built up from (unique) event names and operations: ϵ (nullary), $*$ (unary), and $\cup$, $\circ$, $\|$ (binary).

EXAMPLE 14. Consider EN system N from Example 13. Trace behaviours of its atoms (presented in Fig. 8) are:

$$TB_{N_1} = (D(a,b),[(ab)^*(a \cup \varepsilon)]),$$
$$TB_{N_2} = (D(a,c),[(ac)^*(a \cup \varepsilon)]),$$
$$TB_{N_3} = (D(a,b),[(ab)^*(a \cup \varepsilon)]),$$
$$TB_{N_4} = (D(a,c,d),[(a(c \cup d))^*(a \cup \varepsilon)]).$$

By idempotency of ‖ we get

$$TB_{N_1+N_3} = TB_{N_1} \| TB_{N_3} = (D(a,b),[(ab)^*(a \cup \varepsilon)]).$$

Now, set

$$X = TB_{N_2},\ Y = TB_{N_4},\ X_0 = (D(a,c),[\varepsilon]),\ Y_0 = (D(a,b,c),[\varepsilon]);$$

by Theorem 6 we have the following equations for X and Y:

$$X = [ac]X \cup [a \cup \&]X_0,\ Y = [ac \cup ad]Y \cup [a \cup \varepsilon]Y_0;$$

and by Theorem 7 we get the following equation for X‖Y:

$$\begin{aligned} X\|Y &= [ac]X\|[ac]Y \cup [ac]X\|[ad]Y \cup [ac]X\|[a \cup \varepsilon]Y_0 \cup \\ &\quad [a \cup \varepsilon]X_0\|[ac \cup ad]Y \cup [a \cup \varepsilon]X\|[a \cup \varepsilon]Y_0 \\ &= [ac]X\|[ac]Y \cup [a]X_0\|[ad]Y \cup [a]X_0\|[a]Y_0 \cup [\varepsilon]X_0\|[\varepsilon]Y_0 \end{aligned}$$

(all remaining terms vanish); by Theorem 7(iii) we obtain

$$X\|Y = [ac](X\|Y) \cup [ad](X_0\|Y) \cup [a](X_0\|Y_0) \cup [\varepsilon](X_0\|Y_0);$$

set $Z = X\|Y$, $Z_0 = X_0\|Y_0$; since $X_0\|Y = (D(a,c,d),[\varepsilon]) = Z_0$ we have the following equation:

$$Z = [ac]Z \cup [ad \cup a \cup \varepsilon]Z_0$$

for X‖Y. By Theorem 6 the least solution of this equation is $[(ac)^*(ad \cup a \cup \varepsilon)]Z_0$;

thus,

$$TB_{N_2+N_4} = (D(a,c,d),[(ac)^*(ad \cup a \cup \varepsilon)]).$$

To get TB_N we need to calculate

$$TB_{N_1+N_3} \sharp TB_{N_2+N_4};$$

which has already been done in Example 12; thus, finally

$$TB_N = (\Sigma, [(abc)^*(abd \cup ab \cup ad \cup a \cup \varepsilon)])$$
$$= (\Sigma, Pref([(abc)^*(abd)])),$$

where $\Sigma = D(a,b) \cup D(a,c,d)$. Independent events in Σ are then (b,c) and (b,d). The structure of states of TB_N is given in Fig. 6. □

Notice that the dependency of atomic EN systems is full; it means that in expressions for trace behaviour of atoms the sequential iteration * can be replaced by the trace iteration $^\sharp$. It means that the trace behaviour of any atom is a regular trace system. Since the synchronization operation preserves the trace regularity of systems, we have the following theorem.

THEOREM 10. The trace behaviour of any EN system is a trace regular language. □

Naturally, to find the behaviour of an EN system we do not need to decompose it into atoms; it suffices to decompose it into modules of already known behaviours, e.g. into sequential subnets the behaviour of which can be found by methods elaborated within the theory of sequential systems.

5. TRACES AND PROCESSES

5.1. **State space.** In the last chapter the behaviour of EN systems was expressed in terms of some prefix closed and proper trace systems. In the present chapter concurrent processes viewed as trace languages will be discussed in an abstract framework, i.e. without referring to concrete mechanisms (as e.g. EN systems) generating them. Here, the basic notions will be that of an (atomic) action, represent by a symbol (letter) and that of the dependency / independency of such symbolic actions; let then $\Sigma = (A,D)$ be a concurrent alphabet, fixed for the rest of this chapter, and θ be the set of all traces over Σ. Elements of Σ will represent atomic actions and pairs in D - their mutual dependency. Because of our intended interpretation, traces over Σ will be called <u>states</u>, and the set θ of all states over Σ - the <u>state space</u>; θ is partially ordered by the domination relation $\sqsubseteq$:

$$\sigma_1 \sqsubseteq \sigma_2 \iff \sigma_1 \in Pref(\sigma_2).$$

For any subset Q of θ define

$$\uparrow Q = \{\tau \in \theta \mid \exists \sigma \in Q: \sigma \sqsubseteq \tau\},$$
$$\downarrow Q = \{\tau \in \theta \mid \exists \sigma \in Q: \tau \sqsubseteq \sigma\}.$$

We say that a subset Q of θ dominates another subset R of θ (or that R is dominated by Q), if $R \subseteq \downarrow Q$. Two states are consistent, if there is a state in the state space dominating both of them and inconsistent otherwise. A set P of states is said to be proper, if any two of its consistent states are dominated by a state in P, and directed, if arbitrary two states in P are dominated by a state in P. Clearly, any directed set of states is proper, but not the other way around. A maximal chain in P will be called a line in P. A set of states is conflict-free, if any two of its states are consistent; and it is sequential, if any two its consistent states are comparable. Thus, each sequential set of states is proper.

5.2. **Trace structures.** By a trace structure over Σ we shall mean any prefix closed and proper trace language over Σ; traces in a trace structure T will be called states of T.

From Propositions 14 and 15 it follows that if (Σ,T) is the trace behaviour of any EN system, T is a trace structure over Σ; the convers is not true in general. By the diagram of a trace structure T we shall understand the graph of the poset $(T,\sqsubseteq)$; in such a graph arcs resulting from others by transitivity or reflexivity will be omitted. Examples of such graphs were given in Fig. 5(b) and Fig. 6.

Trace structures are intended to represent the behaviour of concurrent systems. Let T be a trace structure and let σ', σ'' be states in T. As it has been mentioned above, if a state σ' is dominated by another state σ'', it means that there is a run of the represented concurrent system, containing both of them, in which σ'' comes up not earlier than σ'. Thus, two states are comparable, if and only if they appear in the same execution of the concurrent system and the history of one of them is an initial part of the history of the other. If σ' is incomparable with σ'', two cases are possible: either both of them are dominated by a common state (then they are consistent), or such a common state does not exist (then they are inconsistent). In the first case σ' and σ'' are two states of the same run of the concurrent system, but resulting in effect of concurrent actions of the system; they identify different

pieces of the same history, exhibited later by the common dominating state. In the second case σ' and σ'' are states identifying pieces of two different histories, resulting in effect of a choice (a conflict resolution) made earlier during the system action.

A trace structure T is sequential (conflict free), if T is a sequential (conflict free) trace language.

PROPOSITION 12. A trace structure is sequential if and only if its diagram is a rooted tree; a trace structure is conflict free, if and only if it is directed. □

Maximal (w. r. to the inclusion ordering) directed subsets of a trace structure T will be called processes in T. Clearly, all processes are prefix closed. Because of directedness, any process contains only states appearing in a single run of the represented system, since all of them can be viewed as prefixes of the same history. Because of maximality, processes describe full executions of the concurrent system being interpreted.

EXAMPLE 15. It has been shown that the trace behaviour of the EN system from Example 13 is (Σ,T) where

$$\Sigma = D(a,b) \cup D(a,c,d),\ T = Pref[(abc)^*abd];$$

T is a trace structure; the diagram of T has been presented in Fig. 6. Processes in T are languages

$$P_0,\ P_1,\ \ldots,\ P_n,\ \ldots,\ P_\omega,$$

where $P_n = Pref[(abc)^n abd]$ for each $n \in \omega$, and $P_\omega = Pref[(abc)^*]$. Observe that for each n process P_n is finite, and P_ω is infinite.

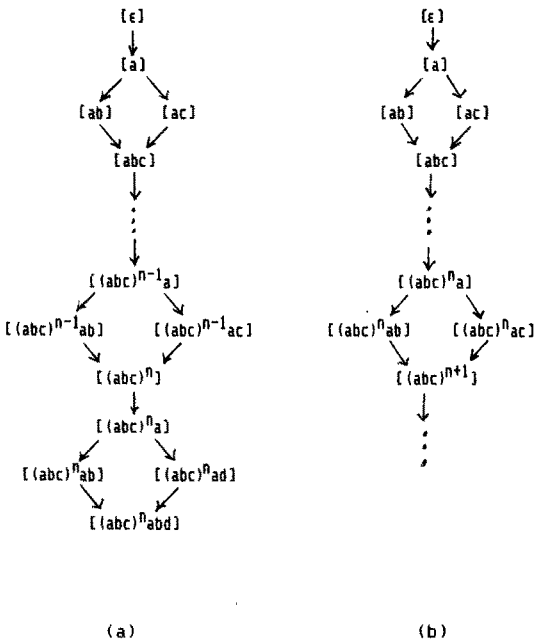
Diagrams of P_n and of P_ω are given in Fig. 6.

PROPOSITION 13. The set of processes of any trace structure is not empty. □

Let T be a trace structure; a state is terminal in T, if it is a maximal state of T; clearly, if a trace structure T contains a terminal state σ, then $Pref(\sigma)$ is a finite process in T; conversely, any finite process in T is equal to $Pref(\sigma)$ for some terminal state σ.

PROPOSITION 14. The only process in a conflict free trace structure is the trace structure itself. □

PROPOSITION 15. A subset of a sequential trace structure T is a process in T if and only if it is a line in T. □



(a) (b)

Fig. 7: Processes P_n (a) and P_ω (b)

5.3. **Observations.** To prove inevitable properties of concurrent systems (discussed in the next section) we need the notion of an observation. Intuitively speaking, an observation is a finite or infinite sequence of states as seen by a sequential observer of a system. Because of possible concurrency, different observers can notice different order of action execution and, consequently, their observations consist of different system states. Objective properties of the system behaviour are independent on particular observations, hence such properties should be discovered by all system observations provided they are faithful, i.e. they do not overlook any action performed by the system. These intuitions give rise to the following formal definition.

By an observation of a trace structure T we mean any line V in T such that $\downarrow V$ is a process in T.

Recall here that any process is a maximal directed subsystem of T, i.e. it contains every possible state of a single run of the represented system; each line is a maximal sequential observation of the system run; to guarantee the "faithfulness" of such an observation we require the maximality of $\downarrow V$. The set $\downarrow V$ of states is obviously directed for any line V in T, but not necessarily it is a maximal directed subsystem of T. Because of this maximality property, one can say that an observation is a line that dominates everything possible to be dominate by a line.

THEOREM 8. For each system T and each process P in T there exists an observation of T dominating P. □

EXAMPLE 16. Let $\Sigma = D(a) \cup D(b)$, $T = \{[a^i b^j] \mid 0 \leq j \leq i < \omega\}$. T is clearly a system. The diagram of T is given in Fig. 7.

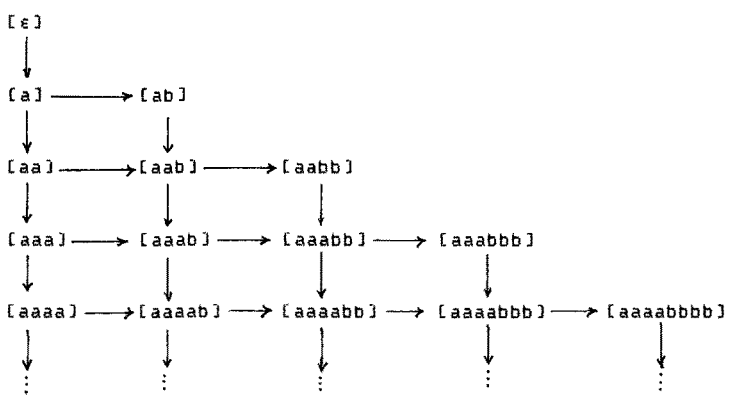


Fig. 7

Observe first that the system is conflict free (all its states are consistent); thus, any observation must dominate the whole system. The line

$$V = ([\varepsilon],[a],[ab],[aba],[abab],\ldots[(ab)^n],[(ab)^n a],\ldots)$$

is an observation of T, since any state in T is dominated by a state in V, e.g. $[abbbaabaaa] = [aaaaaabbbb] \sqsubseteq [aaaaaabbbbbb] = [abababababab] \in V$. However, not all

lines in T are observations; e.g. the line

$$U = ([\varepsilon],[a],[aa],\ldots[a^n],\ldots),$$

is not an observation, since it leaves undominated all states $[a^1b^j]$ with $j>0$.

Observe that in T there is infinitely many lines that are observations of T as well as infinitely many lines that are not such observations. Observe also that each finite chain in T can be extended to an observation of T as well as to a line being not an observation of T. It means that the property of a line "to be an observation" is not finitary: it is not sufficient to know a finite part of a line to decide whether the line is an observation or not. □

Call a trace structure T <u>well synchronized</u>, if for each state σ of T there exists a finite number of states of T incomparable and consistent with σ.

PROPOSITION 16. Every line in a well synchronized trace structure is an observation. □

EXAMPLE 17. The system presented in Fig. 6 is well synchronized; consequently, every line in it is an observation. The system presented in Fig. 7 is not well synchronized: for each $n \in \omega$ state $[a^{n+2}]$ is consistent and incomparable with the state $[ab]$. □

5.4. **Invariancy and inevitability.** Let T be a trace structure over Σ; by a <u>property</u> we shall mean any subset of states in $\theta(\Sigma)$. We say that a property $\mathfrak{Q}$ is <u>invariant</u> in T, if and only if

$$\lambda(T) \cap \uparrow\mathfrak{Q} \subseteq \mathfrak{Q}. \ \square$$

It means that once the system reaches a state with an invariant property, all subsequent states will have this property.

THEOREM 12. A property $\mathfrak{Q}$ is invariant in a trace structure T over (A,D), if for each $\sigma \in \mathfrak{Q}$ and each $a \in A$

$$\sigma[a] \in T \ \Rightarrow \ \sigma[a] \in \mathfrak{Q}. \ \square$$

Thus, checking the invariancy of a property in concurrent systems is relatively simple and similar to that in sequential ones: $\mathfrak{Q}$ is invariant, if for each state in $\mathfrak{Q}$ any possible next state is also in $\mathfrak{Q}$.

PROPOSITION 15. $\emptyset$ is an invariant property; T is an invariant property; if Θ_1 and Θ_2 are invariant properties, then

$$\Theta_1 \cup \Theta_2,\ \Theta_1 \cap \Theta_2$$

are also invariant properties. □

In order to define inevitability in trace structures we have to refer to observations. Intuitively, a property is inevitable in a system, if sooner or later the system reaches a state with this property. Inevitability of a property must be objective, i.e. it must not depend on particular observations of the system; hence, if a property is inevitable, any observer of the system must sooner or later notice a state with this property. The formal definition is as follows.

A property Θ is <u>inevitable</u> in T, if each observation of T intersects Θ (i.e. if $V \cap \Theta \neq \emptyset$ for any observation V of T).

A property that is not inevitable is called <u>avoidable</u>.

EXAMPLE 18. Consider system T from Example 17. The (one state) property Θ_{nm}:

$$\Theta_{nm} = \{[a^n b^m]\},$$

is avoidable in T for any $n,m \in \omega$, since the following line is an observation of T not intersecting Θ_{nm}:

$$[\varepsilon],[a],\ldots[a^{n+1}],[a^{n+1}b],\ldots[a^{n+1}b^m],[a^{n+1}b^m a],\ldots[a^{n+1}b^m(ab)^n],\ldots).$$

The property R_{nm}:

$$R_{nm} = \{[a^i b^j] \mid i \geq n,\ j \geq m\}$$

is inevitable in T for each $n,m \in \omega$, since any observation of T must contain a state dominating $[a^k b^k]$ with $k = \max(n,m)$; such a state clearly belongs to R. □

In general, it is more difficult to prove inevitability than invariancy, mainly because of the nonlocal character of the observation definition. The following proposition gives simple facts about inevitability that can be useful in checking inevitability of some properties.

PROPOSITION 15. T is inevitable in T; [ε] is inevitable in T; if $Q_1 \subseteq Q_2$ and Q_1 is inevitable in T, then Q_2 is inevitable in T; if each process in T contains a state in Q, then ↑Q is inevitable in T. □

Proving invariancy and/or inevitability of some properties (and some combination of them) is of primary interest in analysis and design of concurrent systems. The above formulated notions are thought as some "trace" primitives which can be helpful in creating a partial order model for developing a temporal logic with concurrency.

6. BIBLIOGRAPHICAL NOTES

The motivation for developing the presented theory was to create means for analysing the behaviour of Petri nets as introduced in [Pet1-Pet2], [R1], [RT1]. The notion of traces and of dependency graphs has been introduced in [M1] and presented at Aarhus Workshop on Concurrency in 1977. Since then the theory started to develop in several directions: investigation of algebraic properties [AW1], [BBMS1], [BG1], [BMS1-BMS7], [Kn1-Kn2], [KG], [M4], [Sz1], [T1], [O1], [S1], [Ry1]; connections with graph theory [AR1]; applications to the analysis of concurrent systems [FR], [J1-J4], [Z1-Z2], [Zu1]; comparison with other formalisms [KG1], [M4]. Independently, since 1978 the algebra of free partially commutative monoids started to develop [[CdL1], [CF1], [CL1-CL2], [CM1], [CP1], [D1-D2], [F1], [Me1-Me2], [P1]. Since the application of traces is limited to finite (0,1)-marked nets, some more sophisticated tools has been proposed in [G1], [St1-St2]. Traces introduced in [Re1] correspond to traces presented here but without independency of events.

Elementary net systems considered here has been introduced in [Ro1], [Th1]. *In* [Ro1] an application of traces for the behaviour description is given. The synchronization operation for trace systems has been defined in [M5] with an extensive use of projections due to [GKR1], [KG1]. Modular and fixed point technique for calculating the trace behaviour of nets is given in [M5]. Trace structures are similar to the event structures introduced in [NPW1], but based on dependency / independency relation rather than the conflict relation. The notion of inevitability in the trace structure framework is a particular case of a more general concept discussed in a forthcoming paper.

Acknowledgements. The author gratefully acknowledges the help of G. Rozenberg in

preparation of this paper. Thanks are due to my colleagues E. Ochmański and W. Penczek who were the first readers of the paper. The author is indebted to IJ. J. Aalbersberg for his effort in collecting the bibliography on traces.

7. REFERENCES

[AR1]: **AALBERSBERG, IJ. J., ROZENBERG, G.:** "Traces, dependency graphs and DNLC grammars", Discrete Applied Mathematics 11 (1985) 299 - 306

[AR2]: **AALBERSBERG, IJ. J., ROZENBERG, G.:** "Trace languages defined by context-free languages", manuscript, (1985)

[AW1]: **AALBERSBERG, IJ. J., WELZL, E.:** "Trace languages defined by regular string languages", to appear (1985)

[BBMS1]: **BERTONI, A., BRAMBILLA, M., MAURI, G., SABADINI, N.:** "An application of the theory of free partially commutative monoids: asymptotic densities of trace languages", Lecture Notes in Computer Science 118 (1981) 205 - 215

[BG1]: **BERTONI, A. and GOLDWURM, M.:** "Average analysis of an algorithm for a membership problem of trace languages", manuscript, (1986)

[BMS1]: **BERTONI, A., MAURI, G., SABADINI, N.:** "A hierarchy of regular trace languages and some combinatorial applications", Proc. 2nd World Conf. on Math. at the Serv. of Men, Las Palmas (1982) 146 - 153

[BMS2]: **BERTONI, A., MAURI, G., SABADINI, N.:** "Equivalence and membership problems for regular trace languages", Lecture Notes in Computer Science 140 (1982) 61 - 71

[BMS3]: **BERTONI, A., MAURI, G., SABADINI, N.:** "Context - free trace languages", Proc. 7th CAAP, Lille (1982) 32 - 42

[BMS4]: **BERTONI, A., MAURI, G., SABADINI, N.:** "Equivalence and membership problems for regular and context - free trace languages", manuscript (1982)

[BMS5]: **BERTONI, A., MAURI, G., SABADINI, N.:** "Concurrency and commutativity", manuscript (1982)

[BMS6]: **BERTONI, A., MAURI, G., SABADINI, N.:** "Unambiguous regular trace languages", Proc. Colloq. on Algebra, Combinat. and Comp. Science, Gyor (1983)

[BMS7]: **BERTONI, A., MAURI, G., SABADINI, N.:** "Representation of prefixes of a trace and membership problem for context - free trace languages", manuscript (1985)

[BFP1]: **BEST, E., FERNANDEZ, C. and PLUNNECKE, H.:** "Concurrent systems and processes", manuscript (1985)

[CdL1]: **CARPI, A. and LUCA, A. de:** "Square - free words on partially commutative free monoids", Information Processing Letters 22 (1986) 125 - 132

[CF1]: **CARTIER, P. and FOATA, D.:** "Problemes combinatoires de commutation et rearrangements", Lecture Notes in Mathematics 85 (1969)

[CL1]: **CLERBOUT, M. and LATTEUX, M.:** "Partial commutations and faithful rational transductions", Theoretical Computer Science 34 (1984) 241 - 254

[CL2]: **CLERBOUT, M. and LATTEUX, M.:** "Semi - commutations", Techn. Rep. IT-63-84, Equipe Lilloise d'Inform. Theor., Univers. de Lille 1, Villeneuve d'Ascq. (1984)

[CM1]: **CORI, R. and METIVIER, Y.:** "Recognizable subsets of some partially abelian monoids", Theoretical Computer Science 35 (1984) 179 - 189

[CP1]: **CORI, R. and PERRIN, D.:** "Automates and commutations partielles", RAIRO Informatique Theorique 19 (1985) 21 - 32

[D1]: **DUBOC, C.:** "Some properties of commutation in free partially commutative monoids", Information Processing Letters 20 (1985) 1 - 4

[D2]: **DUBOC, C.:** "Equations in free partially commutative monoids", Lecture Notes in Computer Science 210 (1986) 192 -202

[FR1]: **FLE, M. P. and ROUCAIROL, G.:** "On serializability of iterated transactions", Proc. ACM SIGACT-SIGOPS Symp. on Princ. of Distrib. Comp., Ottawa (1982) 194 - 200

[FR2]: **FLE, M. P. and ROUCAIROL, G.:** "Fair serializability of iterated transactions using FIFO - nets", Lecture Notes in Computer Science 188 (1985) 154 - 168

[FR3]: **FLE, M. P. and ROUCAIROL, G.:** "Maximal serializability of iterated transactions", Theoretical Computer Science 38 (1985) 1 - 16

[F1]: **FOATA, D.:** "Rearrangements of words", in M. Lothaire: Combinatorics on words, Addison-Wesley, Reading, (1983) Chapter 10

[G1]: **GRABOWSKI, J.:** "On partial languages", Fundamenta Informaticae 4 (1981) 427 - 498

[GKR]: **GYORY, G., KNUTH, E., RONYAI, L.:** "Grammatical projections 1. Elementary constructions", Working Paper II/3, MTA SZTAKI, Budapest (1979)

[J1]: **JANICKI, R.:** "Synthesis of concurrent schemes", Lecture Notes in Computer Science 64 (1978) 298 - 307

[J2]: **JANICKI, R.:** "On the design of concurrent systems", Proc. 2nd Intern. Conf. on Distrib. Comp. Systems, Paris (1981) 455 - 466

[J3]: **JANICKI, R.:** "Mazurkiewicz traces semantics for communicating sequential processes", Proc. 5th European Workshop on Appl. and Theory of Petri Nets, Aarhus (1984) 50 - 70

[J4]: **JANICKI, R.:** "Trace semantics for communicating sequential processes", Techn. Rep. R-85-12, Inst. for Elektr. Syst., Aalborg Univ., Aalborg (1985)

[K1]: **KELLER, R. M.:** "A solvable program-schema equivalence problem", Proc. 5th Ann. Princeton Conf. on Inf. Sciences and Systems, Princeton (1971) 301 - 306

[Kn1]: **KNUTH, E.:** "Petri nets and regular trace languages", manuscript (1978)

[Kn2]: **KNUTH, E.:** "Petri nets and trace languages", Proc. 1st Europ. Conf. on Parallel and Distr. Processing, Toulouse (1979) 51 - 56

[KG1]: **KNUTH, E. and GYORY, G.:** "Paths and traces", Computational Linguistics and

Computer Languages 13 (1979) 31 - 42

[L1]: LEVI F. W.: "On semigroups", Bull. of the Calcutta Math. Soc. 36 (1944) 141 - 146

[M1]: MAZURKIEWICZ, A.: "Concurrent program schemes and their interpretations", DAIMI Rep. PB-78, Aarhus Univ., Aarhus (1977)

[M2]: MAZURKIEWICZ, A.: "Equational semantics of concurrent systems", Proc. 8th Spring School on Comp. Sci., Colleville sur mer (1980)

[M3]: MAZURKIEWICZ, A.: "A calculus of execution traces for concurrent systems", manuscript (1983)

[M4]: MAZURKIEWICZ, A.: "Traces, histories, graphs: instances of a process monoid", Lecture Notes in Computer Science 176 (1984) 115 - 133

[M5]: MAZURKIEWICZ, A.: "Semantics of concurrent systems: a modular fixed-point trace approach", Lecture Notes in Computer Science 188 (1985) 353 - 375

[Me1]: METIVIER, Y.: "Une condition suffisante de reconaissabilite dans un monoide partiellement commutatif", Techn. Rep. 8417, U.E.R. de Math. et d'Inform., Univ. de Bordeaux, Bordeaux (1984)

[Me2]: METIVIER, Y.: "Sous-ensemble reconnaisable d'un monoide partiellment commutatif libre", manuscript (1985)

[NPW1]: NIELSEN, M., PLOTKIN, G., WINSKEL, G.: "Petri nets, event structures and domains, Part 1", Theoretical Computer Science 13 (1981) 85-108

[O1]: OCHMANSKI, E.: "Regular trace languages", Ph. D. Thesis (1985) (summary in Bull. of EATCS 27, 1985)

[P1]: PERRIN, D.: "Words over a partially commutative alphabet", NATO ASI Series F12 (1985) 329 - 340

[Pet1]: PETRI, C.A.: "Concepts of Net Theory", Proceedings of the Symposium and Summer School on MFCS'73, High Tatras (1973)

[Pet2]: PETRI, C.A.: "Non-Sequential Processes", GMD-ISF Report 77.05, Gesellschaft fuer Mathematik und Datenverarbeitung mbH Bonn (1977)

[R1]: REISIG, W.: "Petri nets - an introduction", EATCS Monographs on Theoretical Computer Science, Springer Verlag (1985)

[Re1]: REM, M.: "Concurrent Computations and VLSI Circuits", Control Flow and Data Flow: Concepts of Distributed Programming (M. Broy, editor), Springer (1985) 399 - 437

[Ro1]: ROZENBERG, G.: "Behaviour of elementary net systems", this volume (1986)

[RT1]: ROZENBERG, G., THIAGARAJAN, P.S.: "Petri Nets: Basic Notions, Structure and Behaviour", Lecture Notes in Computer Science 224, (1986) 585 - 668

[Ry1]: RYTTER, W.: "Some properties of trace languages", Fundamenta Informaticae 7 (1984) 117 - 127

[S1]: SAKAROVITCH, J.: "On regular trace languages", to appear (1986)

[St1]: STARKE, P. H.: "Processes in Petri Nets", Elektron. Inf. und Kyb. 17 (1981)

[St2]: STARKE, P. H.: "Traces and semiwords", Lecture Notes in Computer Science 208 (1985) 332 - 349

[Sz1]: SZIJARTO, M.: "A classification and closure properties of languages for describing concurrent system behaviours", Fundamenta Informaticae 4 (1981) 531 - 549

[T1]: TARLECKI, A.: "Notes on the implementability of formal languages by concurrent systems", ICS PAS Report 481, Inst. of Comp. Science, Polish Academy of Sciences, Warszawa (1982)

[Th1]: THIAGARAJAN, P.S.: "Elementary net systems", this volume (1986)

[Z1]: ZIELONKA, W.: "Proving assertions about parallel programs by means of traces", ICS PAS Report 424, Inst. of Comp. Science, Polish Academy of Sciences, Warszawa (1980)

[Z2]: ZIELONKA, W.: "The notes on finite state asynchronous automata and trace languages", manuscript (1982)

[Zu1]: ZUIDWEG, H.: "Trace approach to synchronic distances" (manuscript), University of Leiden, Leiden (1986)