CONCURRENT PROGRAM SCHEMES AND THEIR INTERPRETATIONS

by

Antoni Mazurkiewicz*

Datalogisk Afdeling
Matematisk Institut
Århus Universitet

Ny Munkegade
8000 Århus C
Danmark

To be presented at the Århus Workshop on Verification of Parallel Processes, June 13 - 24, 1977, Århus, Denmark. This is a draft version of a paper to be published elsewhere, not for reviewing and further dissemination.

* Permanent address: Institute of Computer Science, P.O.Box 22, 00-901 Warsaw PKiN, Poland.
(since August 1977)

ABSTRACT

Schemes of concurrent programs are considered. The result of a scheme is defined as a set of traces, where each trace is a partially ordered set of symbol occurrences. It is shown that to each scheme corresponds a set of equations determining the result of the scheme; it is shown how these equations can be solved and that the solutions of these equations are regular trace languages. Next, a notion of action systems is introduced; an action consists of its resources and its transformation. Some properties of action systems are shown. Interpretations of schemes are defined as mappings which assign actions to scheme symbols. Interpreted schemes can be regarded as concurrent programs. It is shown how the results of schemes can be lifted (via interpretations) to the results of programs. Some examples of applications the described methods to prove concurrent programs correct are given.

KEY WORDS AND PHRASES

Program models, program schemes, concurrency, Petri nets, parallelism, formal languages, equations, partial order, interpretation, program correctness.

0. INTRODUCTION

In order to make proofs about systems clear and simple, it is convenient to split them into two parts. One part, say "structural", deals only with a general scheme of a system which has to be proved; the second is connected with detailed description of the system components, operation and data structures, decision structures, etc.; let us call it "substantial" part. In this paper an attempt is made to draw a border line between structural and substantial part of proofs about concurrent programs, to find some general methods for carrying out these parts of proofs, and to combine them together.

This approach is not new; there are papers about concurrent programs dealing with such a division in a more or less implicit way [Karp & Miller 1969],[Hoare 1972],[Keller 1973],[Keller 1974], [Ashcroft 1975],[Lauer 1975],[Mazurkiewicz 1975],[Keller 1976], [Owicki & Gries 1976],[Mazurkiewicz 1976]. Here, however, we make this distinction sharp looking for basic primitives of schemes (subjects of structural part of proofs) and systems (subjects of substantial part).

As a mathematical model of concurrent programs suitable for the structural analysis we take Petri Nets [Petri 1962], [Petri 1973], [Petri 1975], [Petri 1976]. An investigation of this aspect of Petri net theory has already been initiated [Lautenbach & Schmid 1974], [Lauer & Campbell 1975], [Keller 1976], [Mazurkiewicz 1976], [Lautenbach 1977]. The notion of a concurrent scheme defined in the paper is derived directly from concepts of net theory. Each concurrent scheme generates a set of processes; our first goal is to describe processes on the same level of abstraction as string of symbols are described in automata theory. However, strings constituting a perfect model of sequential processes are no more available for concurrent ones, so we need another notion. To this end we define traces, which are alternates of strings for concurrent schemes (cf.[Winkowski 1977]). Treating strings as linearly ordered sets of symbol occurrences, traces are such sets but partially ordered only. The algebra of traces used in the paper is similar to that of strings; we have notions of concatenation, language, regular expressions, etc. Having defined traces, we can reduce the investigation of schemes to analysis of trace languages generated by them. Equations in the domain of trace languages are used extensively to that purpose; we follow here the approach of [Blikle 1972], [Blikle 1973].

To deal with "substantial" parts of proofs, we used here an algebra of actions. Roughly speaking, actions are state transformations of some resources of a system. Actions are independent, if they act on disjoint sets of resources. The notion of independency is crucial through the whole paper;

independent actions are allowed to be performed concurrently. Actions form a complete lattice with a composition operation and an independency relation. Similar systems are considered in [Blikle 1971],[Blikle & Mazurkiewicz 1972], but without independency relation. It is not our intention to analyse this algebra in the paper; what we are concerned with is the notion of interpretations of the algebra of traces into the algebra of actions. Such an interpretation is a bridge between schemes and systems; using the terminology of category theory, it is a functor from the category of trace laguages into the category of action systems.

The paper is organized as follows. First, the notion of traces is introduced; it is shown that the ordinary technique is applicable for solving equations defining trace languages. In the second section concurrent schemes are considered. It is shown how to each scheme an equation defining its processes can be formulated and solved. In the third section a notion of action systems is introduced and interpretations of schemes are defined. Some examples serve to illustrate the method.

1. TRACES AND TRACE LANGUAGES

Let V be an alphabet with elements called letters or symbols. Finite sequences of letters are strings over V; the empty string is denoted by $\mathcal{E}$; the set of all strings over V is denoted by V^*. Let I be a binary relation over V called independence generating relation: we say that two symbols a, b in V are independent, if $a \neq b$ and either (a,b) is in I or (b,a) is in I. Anticipating further considerations we can say that letters in V will represent actions to be performed by concurrent systems; independent letters will be interpreted as actions which can be performed concurrently.

Let V and I be fixed for the rest of this section (except examples). Define $\hat{=}_I$ (or simply $\hat{=}$, if I is understood) as the least equivalence relation in V^* satisfying the following condition for all strings w', w" over V^*:

$$(a,b) \text{ is in } I \Rightarrow w'abw" \hat{=} w'baw". \qquad (1.1)$$

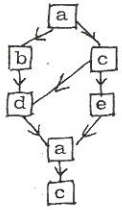
By traces over V w.r. I we shall mean the equivalence classes of $\hat{=}$ relation. Since $\hat{=}$ identifies all strings over V which differ only in the order of consecutive independent letters, in a trace we abstract from this order. This abstraction converts then sequences of symbols, i.e. linearly ordered sets of symbol occurrences, into partially ordered sets of such occurrences.

For any string w over V, the trace containing w will be denoted by $[w]_I$ (or simply [w], if I is understood), and called the trace generated by w. As it is known from the general theory, each string w generates exactly one trace, and any two traces are either identical or disjoint.

EXAMPLE 1.1. Let V = {a,b,c,d,e}, I = {(b,c),(b,e),(d,e)}. The trace [abcdeac] is the set:

{abcdeac, abcedac, acbdeac, acbedac, acebdac}

and can be represented by the graph:



□

Such a construction of a partially ordered set from a family of linearly ordered sets is a particular case of a more general construction which follows from the theorem:

THEOREM 1.1. Any partial ordering relation Q in a set X is uniqely determined by a nonempty family of linear orderings in X (namely, the family of all extensions of Q to linear orderings); conversely, any nonempty family of linear orderings in X determines uniquely a partial ordering in X (namely, the intersection of all members of the family).

Proof follows directly from the Szpilrajn-Marczewski theorem stating that each partial ordering can be extended to a linear ordering.□

Define the concatenation of two traces by the equality:

$$[w][u] = [wu]; \tag{1.2}$$

it can be easily proved that this definition is correct, i.e. that the result of concatenation of traces [w] and [u] does not depend on the choice of "representants" w and u ($\hat{=}$ is a congruence w.r. to concatenation). The concatenation operation is associative:

$$[w]([u][v]) = ([w][u])[v], \tag{1.3}$$

and the trace [ε] is the neutral element for concatenation:

$$[\varepsilon][w] = [w][\varepsilon] = [w]. \tag{1.4}$$

Let L be a set of strings over V; denote by [L] the set of all traces generated by strings in L:

$$[L] = \{[w] | w \text{ is in } L\}. \tag{1.5}$$

From this definition it follows that for any languages L', L" over V (i.e. sets of strings over V) we have

$$L' \subseteq L'' \Rightarrow [L'] \subseteq [L''], \tag{1.6}$$

$$[L' \cup L''] = [L'] \cup [L''], \tag{1.7}$$

and that $[V^*]$ is the set of all traces over V. Subsets of this set are called trace languages over V; to make the distinction between them and usual languages over V more explicit, the latter will be called sometimes string laguages over V.

The concatenation of two trace languages T', T" is defined as usual:

$$T'T'' = \{t't'' \mid t' \text{ is in } T',\ t'' \text{ is in } T''\}. \tag{1.8}$$

By $\emptyset$ we denote the empty trace language, by E the unity language $\{[\varepsilon]\}$ (we shall usually omit braces $\{,\}$ around singletons). If T is a trace language, then T^0 is defined as E, T^{n+1} as T^nT; the iteration of T is defined as usual:

$$T^* = \bigcup_{n=0} T^n. \tag{1.9}$$

From the definition of trace languages it follows

$$\begin{aligned} &[L'][L''] = [L'L''], \\ &[L]^n = [L^n], \\ &[L]^* = [L^*]. \end{aligned} \tag{1.10}$$

In particular, for w in V^*,

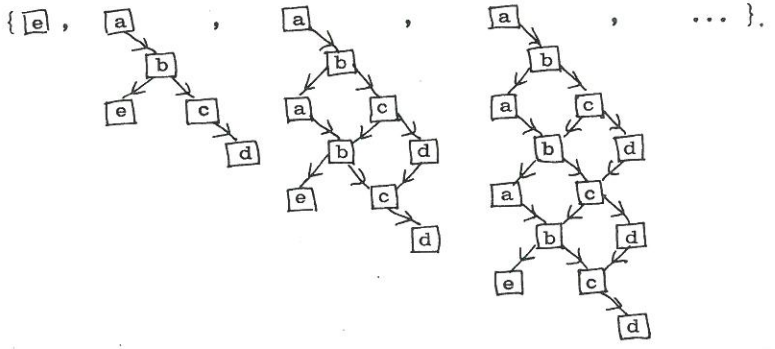
$$\begin{aligned} [w][L] &= \{[w]\}[L] = [wL] = [\{w\}L] \\ &= [\{wu \mid u \text{ is in } L\}], \end{aligned} \tag{1.11}$$

$$\begin{aligned} [w^*] &= [\{\varepsilon, w, ww, www, \ldots\}] \\ &= \{[\varepsilon], [w], [ww], [www], \ldots\}. \end{aligned} \tag{1.12}$$

EXAMPLE 1.2. Let V = {a,b,c,d,e}, I = {(a,c),(a,d),(b,d),(e,c), (e,d)}. Then [(abcd)*e] is the following trace language:

$$\{[e],[abcde],[abcdabcde],[abcdabcdabcde], \ldots\},$$

or, in graphical form:



□

Let A_{ij}, B_i be string languages over V, for i,j = 1,2,...,n. It is known from the theory of formal languages that the set of equations

$$X_i = A_{i1}X_1 \cup A_{i2}X_2 \cup \ldots \cup A_{in}X_n \cup B_i, \quad i = 1,..,n, \tag{1.13}$$

has the least (hence unique) solution being a string language over V. There exists a simple technique which enable us to express this solution by means of iteration, concatenation and union of coefficients of the equation; this technique is based on the fact that the least solution of the equation

$$X = AX \cup B \tag{1.14}$$

is the language A^*B. The same technique can be applied to equations defining trace languages, via the following theorem.

THEOREM 1.2. If the string languages $X_1^0, X_2^0, .., X_n^0$ form the least solution of the set of equations

$$X_i = \bigcup_{j=1}^{n} A_{ij} X_j \cup B_i, \quad i = 1,2,..,n, \tag{1.15}$$

then the trace languages $[X_1^0], [X_2^0], .., [X_n^0]$ form the least solution of the set of equations:

$$X_i = \bigcup_{j=1}^{n} [A_{ij}] X_j \cup [B_i], \quad i = 1,2,..,n. \tag{1.16}$$

Proof. It is known, that

$$X_i^0 = \bigcup_{m=1}^{\infty} X_i^m, \quad i = 1,2,..,n, \tag{1.17}$$

where

$$X_i^1 = B_i, \; X_i^{m+1} = \bigcup_{j=1}^{n} A_{ij} X_j^m \cup B_i. \tag{1.18}$$

Similarly, the least solution of (1.16) is the vector of trace languages $T_1^0, T_2^0, .., T_n^0$, where

$$T_i^0 = \bigcup_{m=1}^{\infty} T_i^m, \; i = 1,2,..,n, \tag{1.19}$$

and where

$$T_i^1 = [B_i], \; T_i^{m+1} = \bigcup_{j=1}^{n} [A_{ij}] T_j^m \; [B_i]. \tag{1.20}$$

But $T_i^1 = [X_i^1]$, and by induction $T_i^m = [X_i^m]$ for each m, hence

$$T_i^0 = \bigcup_{m=0} [X_i^m] = [\bigcup_{m=0} X_i^m] = [X_i^0], \tag{1.21}$$

for each i = 1,2,..,n.□

This theorem allows us to solve equations defining trace languages by means of usual equations for string languages. For instance, the trace language defined in Example 1.2 is the least solution of the equation

$$T = [abcd]T \cup [e]. \tag{1.22}$$

At the end of this section we define the <u>length</u> of any trace over V. Informally, it is the length of the longest path in the graph representing a given trace; formally, the definition is as follows. A string $u = a_1 a_2 \ldots a_n$ is a chain in a string w, if

$$w = w_0 a_1 w_1 a_2 w_2 \cdots w_{n-1} a_n w_n \tag{1.23}$$

for some (possibly empty) strings $w_0, w_1, \ldots, w_n$, and a_i, a_{i+1} are not independent for each i = 1,2,...n-1. The number n is called the length of the chain. Now, the length $|[w]|$ of a trace $[w]$ is defined as the maximal of all lengths of chains in w. It does not depend on the choice of w, since the equivalence $\hat{=}$ preserve the length of the longest chain. It follows directly

from the definition that the following statements are true for all traces t and strings w over V:

$$|t| \geq 0; \tag{1.24}$$

$$|t| = 0 \Rightarrow t = [\varepsilon]; \tag{1.25}$$

$$|t't''| \leq |t'| + |t''|; \tag{1.26}$$

$$|[w]| \leq |w|. \tag{1.27}$$

The last inequality, as we shall see in the next sections, expresses in an abstract way a motivation for organizing tasks in a concurrent systems; the difference $|w| - |[w]|$ is a measure of what we gain due to such an organization.

EXAMPLE 1.3. Let V and I be such as defined in Example 1.2 and let consider the trace $[(abcd)^n e]$ for a positive n. The length of this trace is $2n+2$; the length of the string $(abcd)^n e$ is $4n+1$, so that the gain is here $2n-1$. □

It can be easily observed that string languages are particular cases of trace languages; indeed, if the relation generating independency is empty, then the equivalence relation is reduced to the identity relation, traces are simply strings over V (strictly speaking, one element sets of strings), and the length of traces are exactly the length of corresponding strings.

2. CONCURRENT SCHEMES

Concurrent schemes are thought as abstract models of concurrent systems; a concurrent scheme can be converted into a system by an interpretation which assigns concrete meanings to symbols of actions occurring in the scheme. Therefore, a concurrent scheme corresponds to a class of concurrent systems; facts concerning schemes are then general and valid (via suitable interpretations) for a variety of systems.

By a concurrent scheme Z we shall mean a pair

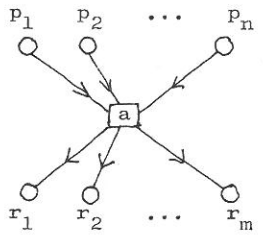
$$(\varphi, S_0)$$

where φ is a function with nonempty finite domain called the set of action symbols of Z; φ assigns to each action symbol an ordered pair of finite nonempty sets called respectively the entry to and the exit from the action symbol; elements of these sets are called control symbols of Z; S_0 is a finite nonempty set of control symbols, called the initial configuration of Z.

Let $Z = (\varphi, S_0)$ be a concurrent scheme fixed for the rest of this section (except examples); all subsequent notions will be defined relatively to this scheme.

Denote the set of action symbols of Z by V, the set of control symbols of Z by P, the entry to $a \in V$ by En(a), the exit from $a \in V$ by Ex(a).

Concurrent schemes can be represented graphically using the graph



to denote the equalities

$$En(a) = \{p_1, p_2, \ldots, p_n\},$$
$$Ex(a) = \{r_1, r_2, \ldots, r_m\},$$

and marking with dots graphic representations of control symbols in S_0.

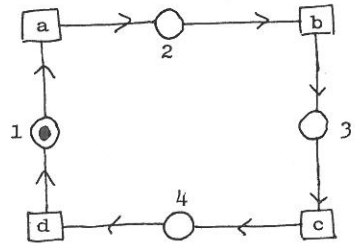
In Example 2.1 we define some simple concurrent schemes which will be considered in the sequel.

EXAMPLE 2.1.

(a) (Four seasons scheme)

$$V = \{a,b,c,d\},\ P = \{1,2,3,4\},\ S_0 = \{1\},$$

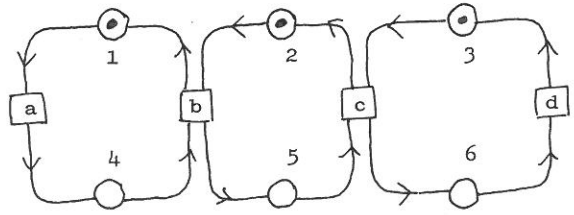
$$En(a) = \{1\} = Ex(d),\ En(b) = \{2\} = Ex(a),$$
$$En(c) = \{3\} = Ex(b),\ En(d) = \{4\} = Ex(c).$$



(b) (Pipeline scheme)

$$V = \{a,b,c,d\},\ P = \{1,2,3,4,5,6\},\ S_0 = \{1,2,3\},$$

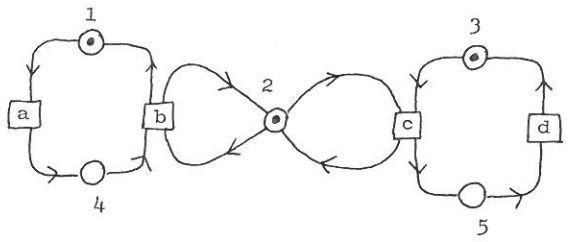
$$En(a) = \{1\},\ Ex(a) = \{4\},\ En(b) = \{2,4\},\ Ex(b) = \{1,5\},$$
$$En(d) = \{6\},\ Ex(d) = \{3\},\ En(c) = \{3,5\},\ Ex(c) = \{2,6\}.$$



(c) (Producer- consumer scheme)

$V = \{a,b,c,d\}$, $P = \{1,2,3,4,5\}$, $S_0 = \{1,2,3\}$,

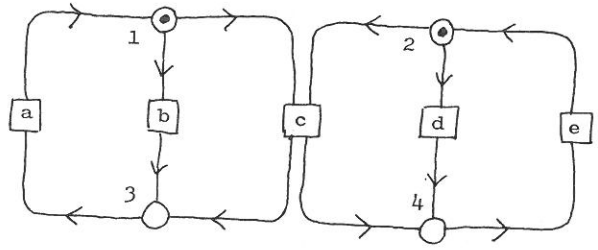
$En(a) = \{1\}$, $Ex(a) = \{4\}$, $En(b) = \{2,4\}$, $Ex(a) = \{1,2\}$,
$En(d) = \{5\}$, $Ex(d) = \{3\}$, $En(c) = \{2,3\}$, $Ex(c) = \{2,5\}$.



(d) (Accident scheme)

$V = \{a,b,c,d\}$ $P = \{1,2,3,4\}$, $S_0 = \{1,2\}$,

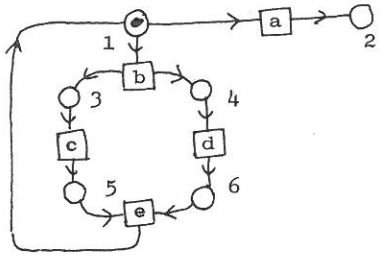
$En(a) = \{3\} = Ex(b)$, $En(d) = \{2\} = Ex(e)$, $En(c) = \{1,2\}$,
$Ex(a) = \{1\} = En(b)$, $Ex(d) = \{4\} = En(e)$, $Ex(c) = \{3,4\}$



(e) (Parallel addition scheme)

$V = \{a,b,c,d,e\}$, $P = \{1,2,3,4,5,6\}$, $S_0 = \{1\}$,

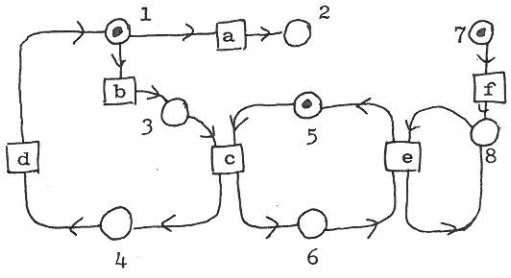
$En(a) = \{1\}$, $Ex(a) = \{2\}$, $En(b) = \{1\}$, $Ex(b) = \{3,4\}$,
$En(c) = \{3\}$, $Ex(c) = \{5\}$, $En(d) = \{4\}$, $Ex(d) = \{6\}$,
$En(e) = \{5,6\}$, $Ex(e) = \{1\}$.



(f)(Parallel factorial scheme)

$V = \{a,b,c,d,e,f\}$, $P = \{1,2,3,4,5,6,7,8\}$, $S_0 = \{1,5,7\}$,

$En(a) = En(b) = Ex(d) = \{1\}$, $Ex(a) = \{2\}$, $En(f) = \{7\}$,
$Ex(b) = \{3\}$, $En(c) = \{3,5\}$, $Ex(c) = \{4,6\}$, $En(d) = \{4\}$,
$En(e) = \{6,8\}$, $Ex(e) = \{5,8\}$, $Ex(f) = \{8\}$.



The scheme Z is said to be <u>proper</u>, if there exists a family C of subsets of P such that:

(i) $S_0 \in C$, (2.1)

(ii) For any subsets S',S" of P:

$S' \subseteq S'' \in C \Rightarrow S' \in C$, (2.2)

(iii) For any subset S of P, and any $a \in V$:

$$\left.\begin{array}{l} En(a) \cap S = \emptyset, \\ En(a) \cup S \in C \end{array}\right\} \Leftrightarrow \left\{\begin{array}{l} Ex(a) \cap S = \emptyset, \\ Ex(a) \cup S \in C. \end{array}\right. \quad (2.3)$$

From now on we shall assume the scheme Z to be proper, and we shall consider only proper schemes in this paper. In particular, all schemes in Example 2.1 are proper.

The family C mentioned in the above definition will be called the family of <u>configurations</u> of Z.

Two configurations S', S" are <u>coexistent</u>, $(S',S'') \in Coex$, if

$$S' \cap S'' = \emptyset,\ S' \cup S'' \in C. \quad (2.4)$$

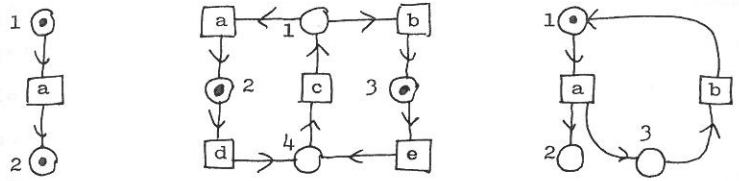
Then (2.3) can be stated as follows:

$$(En(a),S) \in Coex \Leftrightarrow (Ex(a),S) \in Coex. \quad (2.5)$$

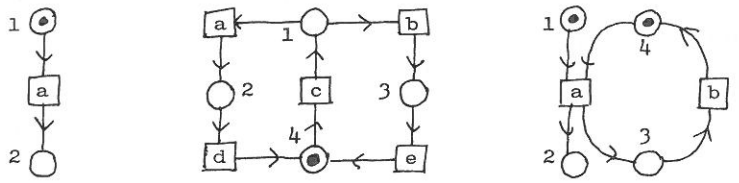
Two action symbols a, b are <u>independent</u>, $(a,b) \in I$, if their entries (or, equivalently, their exits) are coexistent:

$$(En(a),En(b)) \in Coex. \quad (2.6)$$

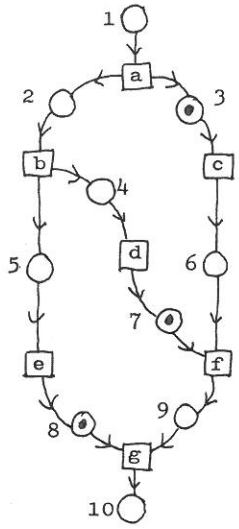
EXAMPLE 2.2. The following schemes are not proper:



but the following are proper:



In the following (proper) scheme:



{1},{3,4},{5,4,6},{5,7,3},{4,3}, {9}
are (some) configurations;

{2,8},{4,7,6},{3,4,9}, {4,9},{2,9}
are not configurations;

({5,7},{3}), ({8},{9}), ({8},{3})
are coexistent;

({5,3},{3,4}), ({5,7},{4,3}),({3},{10})
are not coexistent;

(b,c),(e,c),(e,d),(e,f),(d,c)
are independent action symbols;

(b,f),(a,g),(a,e) are not independent.

The intended meaning of a concurrent scheme is the following. Action symbols correspond to actions of a system components. Configurations correspond to control states of these components; two configurations are coexistent, if they correspond to control states of two disjoint system components, and independent action symbols correspond to actions of such disjoint components. A system is proper, if an action of a component cannot disturb actions of other disjoint components. Consequently, independent action symbols must be interpreted as actions using disjoint resources of the system. In a proper system, conflicts are not excluded; that is, in some configurations a decision should be made which of not disjoint components is allowed to act.

In Example 2.2 some nonproper concurrent schemes are defined as well as examples of configurations, coexistent and not coexistent configurations, independent and not independent action symbols.

In order to define the meaning of concurrent scheme in a strict way, we need some more definitions.

Let $M \subseteq C \times V \times C$ be defined by the equivalence:

$$(S',a,S'') \in M \iff \begin{cases} En(a) \subseteq S', \ Ex(a) \subseteq S'', \\ S'-En(a) = S''- Ex(a). \end{cases} \qquad (2.7)$$

From (2.5) it follows that $(S',a,S'') \in M$ implies

$$(S',S) \in Coex \iff (S'',S) \in Coex. \qquad (2.8)$$

A path from S' to S" in Z is a string of action symbols $a_1a_2\ldots a_n$, $n \geq 0$, such that there is a sequence of configurations:

$$(S_0,S_1,S_2,..,S_n)$$

with

$$S_0 = S', \ S_n = S'', \ (S_{i-1},a_i,S_i) \in M, \qquad (2.9)$$

for all i = 1,2,..,n. Observe that the empty string is a path from S to S for each configuration S.

Each path from S' to S" describe the sequence of system actions while passing from the configuration S' to the configuration S". According to the observations made above, if some action symbols in a path are independent, then the corresponding actions of the system are concurrent, and the order of their execution is not specified. Therefore, if a path has to be a description of what is going on in the real system, we should neglect this order and describe the action between S' and S" by a trace rather than by a string. These traces should be generated from strings over V by the independence relation I in V, like in the previous section.

These considerations lead to the following definition. Let T denotes the set of all traces over V with respect to I. The result of Z is a function

$$Res: C \times C \to T$$

which to each pair of configurations assigns a trace language:

$$Res(S',S'') = \{[w] \mid w \text{ is a path from } S' \text{ to } S''\}. \quad (2.10)$$

The result function of a scheme describes every possible action of a corresponding system - finite as well as infinite, if we agree to understand an infinite string, and an infinite trace, as an infinite set of all its initial finite segments.

Properties of Res function are of our principal interest in this section. In particular, our aim is to reduce investigations of Res function of different schemes to some algebraic operations on trace languages. In Example 2.3 at the end of this section we specify some values of Res function for previously (in Example 2.1) defined schemes, together with proofs of these results; now we formulate some properties of result functions which make possible carrying out such proofs.

The set of all functions from $C \times C$ into T is partially ordered in the obvious way:

$$f' \subseteq f'' \iff \text{for each } S',S'': f'(S',S'') \subseteq f''(S',S''), \quad (2.11)$$

where the sign $\subseteq$ in the right hand side denotes the usual set-theoretical inclusion. Using this ordering we can formulate the following theorem:

THEOREM 2.1. The result function of Z is the least function f from $C \times C$ into T satisfying the following conditions:

$$(i)\ [\varepsilon] \in f(S,S), \quad (2.12)$$

$$(ii)\ (S',a,S'') \in M \Rightarrow [a]f(S'',S) \subseteq f(S',S) \quad (2.13)$$

for all configurations S,S',S".

Proof. It is clear that Res function satisfies (i) and (ii). Let now f satisfies these conditions and let $t \in Res(S',S'')$. Then there is a path $a_1a_2\ldots a_n$ from S' to S" in Z, with $t = [a_1a_2\ldots a_n]$, and a sequence of configurations $(S_0,S_1,S_2,..,S_n)$ such that $(S_{i-1},a_i,S_i) \in M$ for $i = 1,2,..,n$, $S_0 = S'$, $S_n = S''$. Since f satisfies (i),

$$[\varepsilon] \in f(S_n,S_n), \quad (2.14)$$

and since f satisfies (ii),

$$[a_i]f(S_i,S_n) \subseteq f(S_{i-1},S_n),\ i = 1,2,..,n, \quad (2.15)$$

hence

$$[a_1][a_2]\ldots[a_n][\varepsilon] \in f(S_0,S_n), \quad (2.16)$$

i.e. $t \in f(S',S'')$. It means that $Res \subseteq f$; hence Res is the least function satisfying (i) and (ii).□

COROLLARY. The Res function is the least solution of the following set of equations:

$$X(S',S'') = \bigcup_{(S',a,S)\in M} [a]X(S,S''), \text{ if } S' \neq S'', \qquad (2.17)$$

$$X(S',S') = \bigcup_{(S',a,S)\in M} [a]X(S,S') \cup [\varepsilon], \qquad (2.18)$$

where S', S" run over all configurations in C.□

Since the above set of equations is finite (there is only a finite number of all configurations in each scheme), and all these equations are left linear, the values of the solution are regular trace languages (expressible by iteration, union, and concatenation of the coefficients). These values can be found by classical methods; however, the right hand sides of the equations contain usually some redundant terms which introduce nothing but a combinatorial explosion of cases to be considered. To avoid this redundancy, we have the next theorem which makes the solution of the equations much simpler and practically applicable. First, we quote the following lemma:

LEMMA 2.1. If $w' = a_1a_2\cdots a_{i-1}a_i\cdots a_n$ is a path from S' to S" in Z, and $(a_{i-1},a_i)\in I$, then

$$w'' = a_1a_2\cdots a_ia_{i-1}\cdots a_n$$

is also a path from S' to S" in Z, and $[w'] = [w'']$.
Proof is obvious. □

As a matter of fact, the assertion of the above lemma was a motivation for introducing the trace concept.

We say that two configurations S',S" are <u>separate</u> in Z, if they are coexistent and for each a in V:

$$En(a)\subseteq S'\cup S'' \Rightarrow En(a)\subseteq S' \text{ or } En(a)\subseteq S''. \qquad (2.19)$$

THEOREM 2.2. If S',S" are separate configurations, then for each configuration W such that $S' \not\subseteq W$:

$$Res(S'\cup S'',W) = \bigcup_{(S',a,S)\in M} [a]Res(S\cup S'',W) \qquad (2.20)$$

Proof.' $\supseteq$ ' follows from (2.13), since (S',a,S) M implies $(S'\cup S'',a,S\cup S'')$ M. In order to prove '$\subseteq$', we have to show that for each path w from $S'\cup S''$ to W there are $a\in V$, $S\in C$, and a path v from $S\cup S''$ to W such that $(S',a,S)\in M$ and $[w] = [av]$. Let $w = a_1a_2\cdots a_n$. Since $S' \not\subseteq W$, there must occur in this path an action symbol a with $En(a)\cap S' \neq \emptyset$. Let a_i be the first occurrence of such a symbol in w. Since S',S" are separate, $En(a_i)\subseteq S'$, $En(a_k)\subseteq S''$ for $k<i$; hence, $(a_i,a_k)\in I$ for $k<i$. Using (i-1) times Lemma 2.1 we get $[w] = [a_ia_1a_2\cdots a_{i-1}a_{i+1}\cdots a_n]$. Let S be such that $(S',a_i,S)\in M$; then $a_1a_2\cdots a_{i-1}a_{i+1}\cdots a_n$ is a path from S S" to W.□

The above theorem enables us to reduce considerably the amount of work needed to solve equations (2.17),(2.18); it gives, in fact, a possibility to analyse concurrent schemes locally,

dealing with separate configurations. The next theorem gives a method for further simplification of equations which is possible in most frequent cases occurring in practice (e.g. in case of analysing cooperating sequential processes).

We say that an action symbol a is <u>conflict free</u>, if for any other action symbol b, $a \neq b$,

$$En(a) \cap En(b) = \emptyset. \tag{2.21}$$

THEOREM 2.3. If $a \in V$ is conflict free, $En(a) \notin W$, and $(S',a,S'') \in M$, then

$$Res(S',W) = [a]Res(S'',W) \tag{2.22}$$

Proof follows directly from Theorem 2.2.□

Theorems 2.1, 2.2, and 2.3 are basic tools for solving equations for Res function. In Example 2.3 some applications of these theorems for proving facts about schemes are given.

The result function for a scheme can be a basis for some generalizations. E.g. we can extend Res function to <u>Res</u> function:

$$\underline{Res}: 2^C \times 2^C \to T$$

putting

$$\underline{Res}(\underline{S}',\underline{S}'') = \bigcup_{S' \in \underline{S}', S'' \in \underline{S}''} Res(S',S'').$$

By a <u>dead configuration</u> we understand such a configuration S that $En(a) \cap S = \emptyset$ for each a in V. Let D denotes the set of all dead configurations of Z. The function Cont:

$$Cont: C \to T$$

is defined by the equality:

$$Cont(S) = \bigcup_{W \in D} Res(S,W).$$

Equations for Cont have a particularly simple form:

$$Cont(S') = \bigcup_{(S',a,S'') \in M} [a]Cont(S'');$$

if S', S'' are separate, then

$$Cont(S' \cup S'') = \bigcup_{(S',a,S) \in M} [a]Cont(S \cup S'');$$

if $a \in V$ is conflict free, and $(S',a,S'') \in M$, then

$$Cont(S') = [a]Cont(S'').$$

These equations can be used in schemes containing no dead configurations to find Res(S,W) for fixed W; namely, we extend the considered scheme by adding to it a new action symbol c and a new control symbol p, with

$$En(c) = W,\ Ex(c) = \{p\},$$

and then using the equivalence

$$Res(S,W) = L \iff Cont(S) = L[c].$$

EXAMPLE 2.3.

(a) (Four seasons). Denote $Res(\{i\},\{1\})$ by $Q(i)$.

$$Q(1) = [a]Q(2) \cup [\varepsilon] \quad \text{by (2.18)}$$
$$Q(2) = [b]Q(3) \quad \text{by (2.22)}$$
$$Q(3) = [c]Q(4) \quad \text{by (2.22)}$$
$$Q(4) = [d]Q(1) \quad \text{by (2.22)}$$

Hence

$$Q(1) = [abcd]Q(1) \cup [\varepsilon]$$
$$\therefore$$
$$Q(1) = [(abcd)^*].$$

There is no independent symbols; an example of a trace :

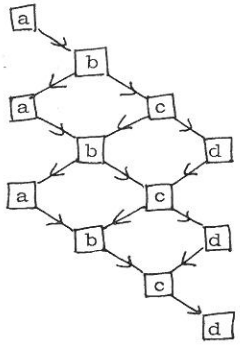
(b) (Pipeline scheme). Denote Res({i,j,k},{1,2,3}) by Q(i,j,k).

$$Q(1,2,3) = [a]Q(4,2,3) \cup [\varepsilon] \quad \text{by (2.18)}$$
$$Q(4,2,3) = [b]Q(1,5,3) \quad \text{by (2.22)}$$
$$Q(1,5,3) = [c]Q(1,2,6) \quad \text{by (2.22)}$$
$$Q(1,2,6) = [d]Q(1,2,3) \quad \text{by (2.22)}$$

Hence

$$Q(1,2,3) = [abcd]Q(1,2,3) \cup [\varepsilon]$$
$$\therefore$$
$$Q(1,2,3) = [(abcd)^*].$$

Independent symbols: (a,c),(a,d),(b,d). Example of a trace:



(cf. Example 1.2)

(c) (Producer - consumer scheme).

Denote Res({i,j,k},{1,2,3}) by Q(i,j,k).

$$Q(1,2,3) = [a]Q(4,2,3) \cup [c]Q(1,2,5) \cup [\varepsilon] \quad \text{by (2.18)}$$
$$Q(1,2,5) = [d]Q(1,2,3) \quad \text{by (2.22)}$$
$$Q(4,2,3) = [c]Q(4,2,5) \cup [b]Q(1,2,3) \quad \text{by (2.17)}$$
$$Q(4,2,5) = [d]Q(4,2,3) \quad \text{by (2.22)}$$

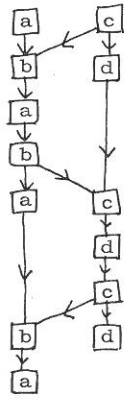
Hence

$$Q(4,2,3) = [cd]Q(4,2,3) \cup [b]Q(1,2,3)$$
$$\therefore$$
$$Q(4,2,3) = [(cd)^*b]Q(1,2,3)$$

Thus

$$Q(1,2,3) = [a(cd)^*b]Q(1,2,3) \cup [cd]Q(1,2,3) \cup [\varepsilon]$$
$$\therefore$$
$$Q(1,2,3) = [(a(cd)^*b \cup cd)^*].$$

Independent symbols: (a,c),(a,d),(b,d). Example of a trace:



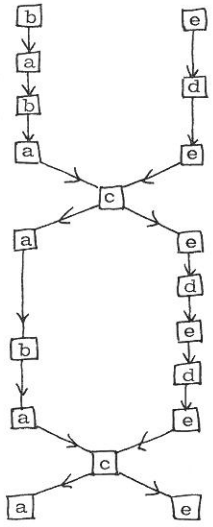
(d)(Accident scheme). Denote Res({i,j},{1,2}) by Q(i,j).

$$Q(1,2) = [b]Q(3,2) \cup [d]Q(1,4) \cup [c]Q(3,4) \cup [\mathcal{E}] \quad \text{by(2.18)}$$
$$Q(3,2) = [a]Q(1,2) \quad \text{by (2.22)}$$
$$Q(1,4) = [e]Q(1,2) \quad \text{by (2.22)}$$
$$Q(3,4) = [a]Q(1,4) \quad \text{by (2.22)}$$

Hence

$$Q(1,2) = [ba \cup de \cup cae]Q(1,2) \cup [\mathcal{E}]$$
$$\therefore$$
$$Q(1,2) = [(ba \cup de \cup cae)^*].$$

Independent symbols: (a,d),(a,e),(b,d),(b,e). Example of a trace:



(e) (Parallel addition scheme)

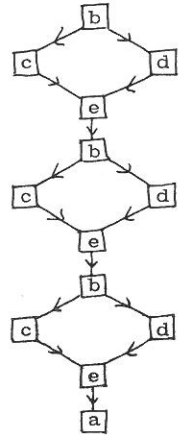
Denote Res({i},{2}) by Q(i).

$$Q(1) = [a]Q(2) \cup [b]Q(3,4) \quad \text{by (2.17)}$$
$$Q(2) = [\mathcal{E}] \quad \text{by (2.18)}$$
$$Q(3,4) = [c]Q(5,4) \quad \text{by (2.22)}$$
$$Q(5,4) = [d]Q(5,6) \quad \text{by (2.22)}$$
$$Q(5,6) = [e]Q(1) \quad \text{by (2.22)}$$

Hence

$$Q(1) = [bcde]Q(1) \cup [a]$$
$$\therefore$$
$$Q(1) = [(bcde)^*a].$$

Independent symbols: (c,d). Example of a trace:



(f) (Parallel factorial scheme)

Denote Res({i,j,k},{2,5,8}) by Q(i,j,k).

$$Q(1,5,7) = [f]Q(1,5,8) \quad \text{by (2.22)}$$
$$Q(1,5,8) = [a]Q(2,5,8) \cup [b]Q(3,5,8) \quad \text{by (2.17)}$$
$$Q(2,5,8) = [\varepsilon] \quad \text{by (2.17)}$$
$$Q(3,5,8) = [c]Q(4,6,8) \quad \text{by (2.22)}$$
$$Q(4,6,8) = [d]Q(1,6,8) \quad \text{by (2.22)}$$
$$Q(1,6,8) = [e]Q(1,5,8) \quad \text{by (2.22)}$$

Hence

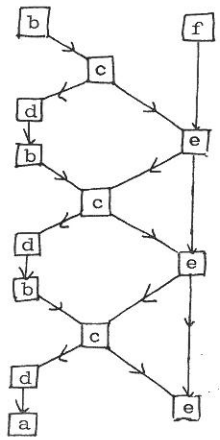
$$Q(1,5,8) = [bcde\,]Q(1,5,8) \cup [a]$$

$\therefore$

$$Q(1,5,8) = [(bcde\,)^*a], \text{ and } Q(1,5,7) = [f(bcde\,)^*a].$$

Independent symbols: (a,e),(a,f),(c,f),(b,e),(b,f),(d,f);

Example of a trace:



3. CONCURRENT SYSTEMS AND INTERPRETATIONS

In this section concurrent systems will be considered. By a concurrent system we shall mean here a concurrent scheme together with an interpretation of its action symbols in a domain of actions, so first we define an action system which will serve as a domain of interpretations.

By an action system , fixed for the rest of this section, we shall mean a pair

$$(R,U) \qquad (3.1)$$

where R is a set (of resources), U is a set (of resource state values). By a state of R we shall mean any mapping

$$s: R \to U;$$

the set of all states of R will be denoted by Σ. Let A denotes the action system (3.1).

By an action in A we mean any pair

$$\langle X,r \rangle,$$

where $X \subseteq R$ is a set of resources called the scope of $\langle X,r\rangle$, and r is a binary relation over Σ:

$$r \subseteq \Sigma \times \Sigma,$$

called the transformation of $\langle X,r\rangle$, such that

$$(s',s'') \in r \Rightarrow \forall x \in R-X: s'(x) = s''(x). \qquad (3.2)$$

The intended meaning of the above condition is that an action does not change any state of resource outside its scope; hence its transformation can be defined only for its scope.

The set of all actions of A will be denoted by F. The set F is partially ordered in the obvious way:

$$\langle X',r'\rangle \subseteq \langle X'',r''\rangle \iff X' \subseteq X'',\ r' \subseteq r''. \qquad (3.3)$$

The action $\langle \emptyset,\emptyset\rangle$ is the least element of F. All actions with empty transformation: $\langle X,\emptyset\rangle$, are called "impossible" actions. Actions can be composed; the composition operation in F is denoted by $\circ$ and defined by the equality:

$$\langle X',r'\rangle \circ \langle X'',r''\rangle = \langle X' \cup X'', r'or''\rangle \qquad (3.4)$$

where r'or" denotes the usual composition of binary relations:

$$(s',s'') \in r'or'' \iff \exists s: (s',s) \in r', (s,s'') \in r''. \qquad (3.5)$$

It is clear that the composition of any two actions is an action again.

Let $\underline{e}$ be the identity relation in Σ, i.e. $(s',s'') \in e \iff s' = s'' \in \Sigma$. Then the action

$$\underline{e} = \langle \emptyset, e\rangle \qquad (3.6)$$

is called the "empty" or "do nothing" action. Observe that in accordance to our intuition "do nothing" action needs no resources to be performed.

Actions can be independent; the independency relation Ind is a binary relation in F defined by the equivalence:

$$(\langle X',r'\rangle, \langle X'',r''\rangle) \in Ind \iff X' \cap X'' = \emptyset. \qquad (3.7)$$

The intended meaning of independency of two actions is that they can be performed concurrently; therefore, two actions can be performed concurrently if and only if they use no common resource.

The notion of independency is basic for the concept of interpretation, as we shall see in the sequel; namely, we shall require that any interpretation preserves independency: independent actions will correspond to independent action symbols.

The following theorem gives properties of action systems.

THEOREM 3.1. Let A be an action system as defined above. Then:

(i) $(F, \subseteq)$ is a complete lattice;

(ii) $(F, o, \underline{e})$ is a monoid;

(iii) For any $f \in F$, $G \subseteq F$:

$$f o(\bigcup G) = \bigcup(foG),$$
$$(\bigcup G) of = \bigcup(Gof),$$

where $\bigcup G$ is the l.u.b. of G,
$foG = \{fog \mid g \in G\}$, $Gof = \{gof \mid g \in G\}$;

(iv) For any f, g in F, $G \subseteq F$:

$$\begin{aligned} &(f,g) \in \mathrm{Ind} \Rightarrow (g,f) \in \mathrm{Ind}, \\ &(f,\underline{e}) \in \mathrm{Ind}, \\ &(f,g) \in \mathrm{Ind} \Rightarrow fog = gof \text{ (but not conversely)}, \\ &(f,G) \in \mathrm{Ind} \Leftrightarrow (f, \bigcup G) \in \mathrm{Ind}, \end{aligned} \tag{3.8}$$

where $(f,G) \in \mathrm{Ind}$ means that for each $g \in G$ $(f,g) \in \mathrm{Ind}$.

Proof is obvious. □

The mentioned above properties of action systems could be taken as axioms characterizing such a systems; in fact, to construct interpretations we need only these properties.

By a <u>concurrent system</u> we understand a pair

$$(Z, \psi) \tag{3.9}$$

where Z is a concurrent scheme (the <u>scheme</u> of the system), as defined in Section 2, and ψ is a mapping of the set of action symbols of Z into the set of actions of an action system (the <u>interpretation</u>). It is assumed that

$$(a,b) \in I \Rightarrow (\psi a, \psi b) \in \mathrm{Ind} \tag{3.10}$$

for each action symbols a, b, where Ind is the independence relation of the action system defined by the interpretation. We shall assume in the sequel that this action system is such as defined above, with F as the set of actions.

In Example 3.1 we give some concurrent systems with schemes defined in previous examples.

Let C be the set of all configurations of Z, V the set of all action symbols of Z. Let w be a path from S' to S" in Z, $w = a_1 a_2 \ldots a_n$, $n>0$. We say that f is the <u>effect</u> of the path w in the system (Z, ψ), if

$$f = \begin{cases} (\psi a_1) o (\psi a_2) o \ldots o (\psi a_n), & \text{if } n>0, \\ \underline{e}, & \text{if } n = 0. \end{cases} \tag{3.11}$$

Let $E_\psi(S',S")$ denotes the set of effects of all paths from S' to S" in (Z,ψ).

The result of a system (Z,ψ) is defined as a function Res_ψ from $C \times C$ into F defined by the equality:

$$Res_\psi(S',S") = \bigcup\{f \mid f \in E_\psi(S',S")\}. \tag{3.12}$$

The value of $Res_\psi(S',S")$ is the total action which is performed during all possible runs of the system from the configuration S' to the configuration S".

Let us extend the interpretation ψ to the mapping ψ^*: V*->F putting

$$\begin{aligned} \psi^*\varepsilon &= \underline{e}, \\ \psi^*a &= \psi a, \quad \text{for } a \in V, \\ \psi^*w'w" &= (\psi^*w')\circ(\psi^*w"), \text{ for } w',w" \text{ in } V^* \end{aligned} \tag{3.13}$$

In the sequel we shall denote ψ and its extension by the same letter ψ.

LEMMA 3.1. For any strings w', w" over V:

$$[w'] = [w"] \Rightarrow \psi w' = \psi w" \tag{3.14}$$

Proof follows from (1.1),(3.8), and (3.10).[]

Extend now ψ to the set of all traces over V putting

$$\psi[w] = \psi w, \quad \text{for each } w \text{ in } V^*; \tag{3.15}$$

this definition is correct since by Lemma 3.1 the value of ψ does not depend on the choice of "representant" w of [w]. Extend now ψ to the set of all trace languages over V defining

$$\psi T = \bigcup\{\psi t \mid t \in T\}, \tag{3.16}$$

for each $T \subseteq [V^*]$.

THEOREM 3.2. For any configurations S',S" of the system (Z,ψ):

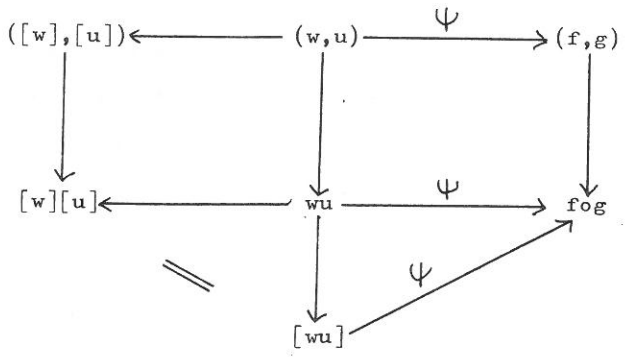
$$\psi Res(S',S") = Res_\psi(S',S"). \tag{3.17}$$

Proof follows from (3.11), (3.12), and from the construction of the extension of ψ. []

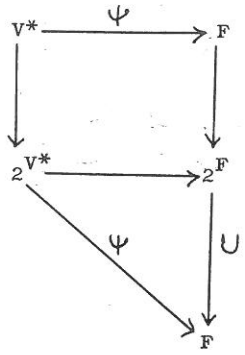
The above theorem states that there is no difference between the result of an interpreted scheme, i.e. of a system, and the interpretation of the result of the scheme; in both cases the effect is the same. Due to this theorem we have a quick method for finding results of concurrent systems; having once determined the result of a scheme, one can easily find the results of any of its interpretations.

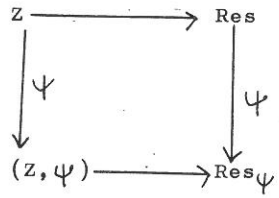
Applications of Theorem 3.2 are given in Example 3.1.

Let us recapitulate in a graphical form the results of this section.



Connections between strings, traces and actions.





Theorem 3.2.

Extensions of interpretations.

All diagrams above commute.

EXAMPLE 3.1. First of all, we introduce a convention to shorten definitions of actions. Let A = (R,U) be an action system with $\sum$ as the set of all states of R. For any binary predicate p let

$$r: p(s',s'')$$

denote

$$r = \{(s',s'') \mid s',s'' \in \sum , p(s',s'')\}.$$

Now, we consider interpretations of schemes (f), (e), and (c).

(f) (Parallel factorial scheme). Let A = (R,U) be an action system with

$$R = \{x,y,z,u\}, \; U = \{\ldots,-1,0,1,2,\ldots\}.$$

Define the following actions:

$x \leq 0 = \langle\{x\},r_1\rangle$, r_1: $s'(x) \leq 0, s''(x) = s'(x)$;
$x > 0 = \langle\{x\},r_2\rangle$, r_2: $s'(x) > 0$, $s''(x) = s'(x)$;
$y:=x = \langle\{x,y\},r_3\rangle$, r_3: $s'(x) = s''(x) = s''(y)$;
$x:=x-1 = \langle\{x\},r_4\rangle$, r_4: $s''(x) = s'(x) - 1$;
$z:=z*y = \langle\{z,y\},r_5\rangle$, r_5: $s''(z) = s'(x)*s'(z)$, $s''(y) = s'(y)$;
$z:=1 = \langle\{z\},r_6\rangle$, r_6: $s''(z) = 1$.

The mapping

ψa = x≤0

ψb = x>0

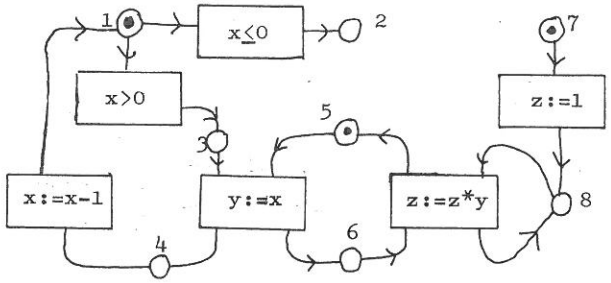
ψc = y:=x

ψd = x:=x-1

ψe = z:=z*y

ψf = z:=1

is then an interpretation, since it preserves independency relation: independent actions correspond to independent action symbols. The system (Z,ψ) can be represented graphically:



Since Res({1,5,7},{2,5,8}) = [f(bcde)*a], by Theorem 3.2 we get

$Res_\psi(\{1,5,7\},\{2,5,8\})$

$= z:=1 \circ (x>0 \circ y:=x \circ x:=x-1 \circ z:=z*y)^* \circ x\leq 0$

$= \langle \{x,y,z\},\ r_6 \circ (r_2 \circ r_3 \circ r_4 \circ r_5)^* \circ r_1 \rangle$

$= \langle \{x,y,z\},\ r_7 \rangle$

where

r_7: s'(x)>0, s"(x) = s"(y)-1 = 0, s"(z) = factorial(s'(x))

or

s'(x)≤0, s"(x) = s'(x), s"(y) = s'(y), s"(z) = 1;

In more familiar form we can write

$Res_\psi(\{1,5,7\},\{2,5,8\})$ =

(x,y,z) := <u>if</u> x>0 <u>then</u> (0,1,factorial(x))

<u>else</u> (x,y,1).

(e)(Parallel addition scheme). We take the same action system as in the previous case, and adopt the same definitions of x>0, x≤0, x:=x-1, but we add one action more:

y:=y+1 = $\langle \{y\}, r_8 \rangle$, r_8: s"(y) = s'(y) + 1.

Then the mapping

ψa = x≤0

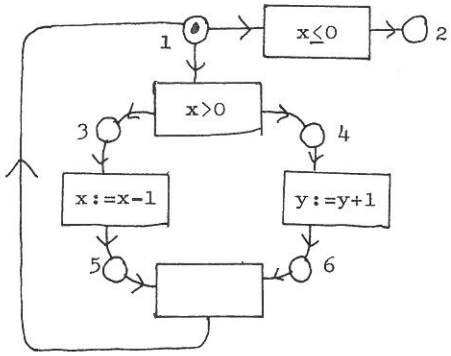
ψb = x>0

ψc = x:=x-1

ψd = y:=y+1

ψe = "do nothing"

is an interpretation of the considered scheme, since $(\psi c, \psi d)$ are independent; such an interpreted scheme can be represented by the graph:



Since Res({1},{2}) = [(bcde)*a], we have

$$Res_{\psi}(\{1\},\{2\}) = (x>0 \circ x:=x-1 \circ y:=y+1)^* \circ x\leq 0$$
$$= \langle\{x,y\}, r_9\rangle$$

where $r_9 = (r_2 \circ r_4 \circ r_8)^* \circ r_1$ i.e.

r_9: s'(x)$\leq$ 0, s"(x) = s'(x), s"(y) = s'(y)
or
s'(x)>0, s"(x) = 0, s"(y) = s'(x) + s'(y);

in more readable form:

$Res_{\psi}(\{1\},\{2\})$ = (x,y) := <u>if</u> x>0 <u>then</u> (0,x+y) <u>else</u> (x,y).

(c) (Producer - consumer scheme). Let now the action system be defined as A = (R,U) with

R = {input, output, queue, x, y},
U = V*, where V is an arbitrary alphabet.

Define the following actions:

read x = $\langle\{input,x\},r_1\rangle$,
r_1: s'(input) = $a_1a_2\ldots a_n$, n>0,
s"(input) = $a_2\ldots a_n$,
s"(x) = a_1;

x to queue = $\langle\{queue,x\},r_2\rangle$,
r_2: s'(queue) = $a_1\ldots a_n$, s'(x) = a,
s"(queue) = $a_1\ldots a_n a$;

y from queue = $\langle\{queue,y\},r_3\rangle$,
r_3: s'(queue) = $a_1a_2\ldots a_n$, n>0,
s"(queue) = $a_2\ldots a_n$
s"(y) = a_1;

write y = $\langle\{output,y\},r_4\rangle$,
r_4: s'(output) = $a_1\ldots a_n$, s'(y) = a,
s"(output) = $a_1\ldots a_n a$;

for arbitrary symbols $a,a_1,a_2,\ldots a_n$ in V.

Then the mapping:

ψa = read x
ψb = x to queue
ψc = y from queue
ψd = write y

is an interpretation of the considered scheme, since (ψa, ψc), (ψa, ψd), (ψb, ψd) are independent. Hence

$$\mathrm{Res}_{\psi}(\{1,2,3\},\{1,2,3\}) = \langle\{\text{input},x,\text{queue},y,\text{output}\},r_5\rangle$$

where $r_5 = (r_1 \circ (r_3 \circ r_4)^* \circ r_2 \cup r_3 \circ r_4)^*$.

We shall not look for an explicit form of r_5 (it is not a function); instead, we prove that

$$r_5 \subseteq r$$

where

r: s'(output)s'(queue)s'(input)
= s"(output)s"(queue)s"(input),

i.e. that the concatenation of output state, queue state, and input state is not changed while passing from the configuration {1,2,3} to {1,2,3} in the considered system. In other words, this concatenation is an invariant for the configuration {1,2,3} in the system.

The proof is as follows. First, check

(1) $r_1 \circ r \circ r_2 \subseteq r$,
(2) $r_3 \circ r_4 \subseteq r$,
(3) $r^* = r$,

Next, we have

(4) $(r_3 \circ r_4)^* \subseteq r^* = r$,
(5) $r_1 \circ (r_3 \circ r_4)^* \circ r_2 \subseteq r$ by (1) and (4),
(6) $(r_1 \circ (r_3 \circ r_4)^* \circ r_2 \cup r_3 \circ r_4)^* \subseteq r^* = r$ by (2), (5).

Therefore, $\mathrm{Res}_{\psi}(\{1,2,3\},\{1,2,3\}) \subseteq r$. □

ACKNOWLEDGEMENTS

The author wish to thank Dr Hartmann J. Genrich and Dr Kurt Lautenbach for valuable discussions, comments and criticism. The author is greatly indebted to Grethe Mayoh for a technical assistance during preparing this paper.

REFERENCES

[Ashcroft 1975] Ashcroft, E. A.: Proving assertions about parallel programs, J.Comp. Sys.Sci. 10, 1 , (1975), 110-135

[Blikle 1971] Blikle,A.: Nets, complete lattices with a composition, Bull. Acad. Polon. Sci.,Ser.Sci.Mathemat. Astronom. Phys. 19 (1971), 1123-1127

[Blikle 1972] Blikle,A.: Equational Languages, Inf.&Contr. 21 (1972), 134-147

[Blikle & Mazurkiewicz 1972] Blikle,A.,Mazurkiewicz,A.:An algebraic approach to the theory of programs, algorithms, languages and recursiveness, Proc. Symp. MFCS'72, Warsaw-Jablonna 1972.

[Blikle 1973] Blikle,A.:Equations in nets-computer oriented lattices, CC PAS Reports 99 (1973)

[Hoare 1972] Hoare,C.A.R.:Towards a Theory of Parallel Programming, Hoare & Perrot, (ed.)Operating Systems Techniques, Acad.Press,(1972)

[Karp & Miller 1969] Karp,R.M.,Miller,R.E.: Parallel Program Schemata, J.Comp.Sys. Sci. 3, 2, (1969), 147-195

[Keller 1973] Keller,R.M.: Parallel Program Schemata and Maximal Parallelism, J.ACM. 20, 3,(1973) 514-537 and 20, 4, (1973) 696-710.

[Keller 1974] Keller,R.M.: Vector Replacement Systems: A Formalism for Modeling Asynchronous Systems, Techn.Report 117, CSL, Princeton Univerisity (1974)

[Keller 1976] Keller,R.M.:Formal Verification of Parallel Programs, Comm.ACM 19, 7, (1976), 371-384

[Lauer 1975] Lauer,P.E.: A Project to Investigate a Design Technique for Asynchronous Systems of Processes, Univ. of Newcastle upon Tyne Memo (1975)

[Lauer & Campbell 1975] Lauer,P.E.,Campbell,R.H.: Formal Semantics for a Class of High-Level Primitives for Coordinating Concurrent Processes, Acta Inf. 5, (1975) 247-332

[Lautenbach & Schmid 1974] Lautenbach,K.,Schmid,H.A.: Use of Petri Nets for Proving Correctness of Concurrent Process Systems, Proc. IFIP'74 Congress, (1974)

[Lautenbach 1977] Lautenbach,K.: Use of Invariants of the Special Net Theory, Proc. of the third Winter School of the Hungarian Academy of Sciences, Visehrad, January 1977

[Mazurkiewicz 1975] Mazurkiewicz,A.: Parallel Recursive Program Schemes, Lecture Notes in Computer Sci., 32. Proc. of MFCS'75, (1975), 75-87

[Mazurkiewicz 1976] Mazurkiewicz,A.: Invariants of Concurrent Programs, Proc. IFIP--INFOPOL Conf., March 1976 (next published in Proc. IFIP-INFOPOL Conf. on Information Processing, North Holland Publ. Co.,1977, 353-372)

[Owicki & Gries 1976] Owicki,S.,Gries,D.: An Axiomatic Proof Technique for Parallel Programs, I , Acta Informatica 6, 4, (1976), 319-340

[Petri 1962] Petri,C.A.: Kommunikation mit Automaten, Diss. TH Darmstadt 1962, Math.Inst. der Universität Bonn (1962)

[Petri 1973] Petri C.A.: Concepts of Net Theory, MFCS'73 Proceedings, High Tatras, Math.Inst. of Slovak Acad. of Sci., (1973),137-146

[Petri 1975] Petri.C.A.: Interpretations of Net Theory, ISF Report 75-207, GMD Bonn, (1975)

[Petri 1976] Petri,C.A.: Nichtsequentielle Prozesse, ISF Report 76-6, GMD Bonn, (1976)

[Winkowski 1977] Winkowski,J.: Algebras of Arrays -- a Tool to Deal with Concurrency, CC PAS Reports 287 (1977)