MATHEMATICAL PROBLEMS IN COMPUTATION THEORY
BANACH CENTER PUBLICATIONS, VOLUME 21
PWN – POLISH SCIENTIFIC PUBLISHERS
WARSAW 1988

TRANSITION SYSTEMS AND CONCURRENT PROCESSES

ANDRÉ ARNOLD

Université de Bordeaux I
Unité associée au CNRS 226, France

Transition systems are a good formalism to describe concurrent processes. In this paper we define this formalism and we present some definitions and properties about transition systems which are relevant when they are used as a theoretical model for studying semantics of concurrent processes.

Introduction

Transition systems play a fundamental role for describing and studying concurrent processes. Indeed a simple, and already introduced way [7] of formalizing the notion of process is to consider a process as a set of *states* together with labelled *transitions* between states describing the actions the process can perform in the different states and the results of these actions. Moreover, it is possible to give a special meaning to some states or transitions, for example some states can be specified as initial or terminal.

Transition systems are now frequently used to describe processes. The language Estelle of description of protocols is based upon this notion and *CCS*-like languages use it for defining their semantics [9].

Some properties of processes which are of interest when studying their behaviours can be easily expressed in the formalism of transition systems, for example, "fair" behaviours are just a specified subset of the set of all possible infinite behaviours and the theory of automata recognizing infinite words [3] allows such specifications; properties of states such as to be deadlocking have a simple characterization in the corresponding transition system. Also usual operations on processes have their counterpart in the formalism of transition systems. In particular, interactions between concurrent processes can be represented by the set of actions which can be simultaneously

performed, and then the whole system is represented by a kind of direct product of its components.

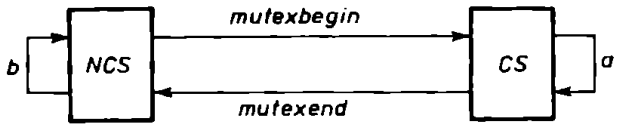
Properties of processes are, in most of the cases, expressed in terms of existence, in the associated transition system, of states satisfying some given properties. We shall show that it is very simple to express a wide family of properties of states of a given transition system, and that it is also very simple to "compute" the set of states of a given transition system satisfying properties of this family.

This paper is divided in four parts. In part 1 we give the definitions of a transition system and of its computations. In Part 2 we explain how some properties of transition systems can be expressed by considering sets of states. In Part 3 we define the transition system associated with a system of interacting processes from the transition systems describing each process in the system and from a formal definition of the interactions taking place in the system. Part 4 is an extension of Part 2 and explains in greater details how sets of states, and also sets of transitions, can be characterized in a transition systems.

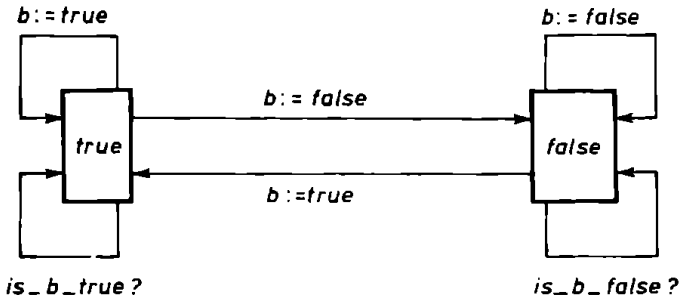
1. Definition of transition systems

1.1. Transition systems. A transition system over an alphabet A of actions is a pair (Q, T), where Q is a set of states and T, included in $Q \times A \times Q$, is a set of labelled transitions. The states of Q are intended to describe the states of the process (values of local variables, program counter,...); the set A is the set of actions the process can perform and a transition $(q, a, q') \in T$ means that in state q, the process can perform the action a, and if it does so, it goes into state q'. The nondeterminism of the system (T can contain transitions (q, a, q_1) and (q, a, q_2) with $q_1 \neq q_2$) amounts to saying that the result of an action is not fully determined by the state of the process when the action is performed and may depend on some other features not taken into account in the model. Here are examples of transitions systems.

EXAMPLE 1. A process with a critical section can be roughly considered as a process with two states: critical section (CS) and noncritical section (NCS) and four actions: the usual *mutexbegin* and *mutexend*, and actions a and b respectively in critical and noncritical sections, described by the following ts:



EXAMPLE 2. A boolean variable b is a process with two states: *true* and *false* and four actions: $b:=true$, $b:=false$, $is\_b\_true\_?$, $is\_b\_false\_?$ described by the following ts:



1.2. Computations. A finite or infinite *computation* of a ts is a sequence $t_1, t_2, t_3, \ldots, t_n, \ldots$ of transitions, such that for every i, if $t_i = (q_i, a, q_i')$ and $t_{i+1} = (q_{i+1}, b, q_{i+1}')$, then $q_i' = q_{i+1}$. We denote by T^* the set of finite computations, by T^ω the set of infinite ones and by T^∞ their union. Indeed we are not interested in every computation but in those starting in a distinguished subset I of Q, the *initial* states and we get the set B^{Init} of initial finite behaviours. It is clear that B^{Init} is closed under left factors, i.e.,

$$B^{Init} = LF(B^{Init}),$$

where

$$LF(L) = \{u/\text{there exists } v \text{ such that } uv \in L\}.$$

Also we can distinguish another subset F of Q and we define the subset B^{Fin} of B^{Init} consisting of computations ending in a state of F, called *terminal computations*, for if the process halts in a state of F, even if it still may perform some actions, one can consider it as normally terminated. On the other hand, if the process is in a state where no action can be performed, and if this state is not in F, then the process is terminated but in an abnormal way (a deadlock for example) and the corresponding computation is in B^{Init}, but not in B^{Fin}.

Finally, we want to distinguish a subset B^{Inf} of T^ω which will be the set of admissible infinite computations, for in the presence of some constraints on infinite computations, for example fairness conditions, not any infinite computation is admissible. We only require that $LF(B^{Inf})$ is included in B^{Init}, which means that every finite prefix of an admissible infinite computation is an initial one, and we mention that the converse is not always true: if an infinite computation has all its prefixes in B^{Init}, it is not necessarily an admissible one (we shall see an example next). This subset can be distinguished by specifying a subset R of Q and defining the admissible infinite

computations as being those which goes infinitely often through a state in R, exactly in the same way one defines infinite words recognized by Buchi automata [3].

EXAMPLE 3. It is usual to assume that a process does not stay forever in its critical section. Therefore in example 1, one can demand that admissible infinite computations are those in which every *mutexbegin* is eventually followed by a *mutexend*. This can be done in defining R to be the set containing only the state *NCS*. ∎

1.3. Processes. Thus we can consider that a process can be represented by a transition system, i.e., a pair (Q, T) together with three sets of computations: B^{Init}, B^{Fin}, B^{Inf} such that:

(i) B^{Init} and B^{Fin} are included in T^*, B^{Inf} in T^ω;

(ii) B^{Fin} is included in $B^{\text{Init}} = LF(B^{\text{Init}})$;

(iii) $LF(B^{\text{Inf}})$ is included in B^{Init}.

These three subsets can be specified by three subsets I, F, R of Q, as explained above.

Condition (iii) can be also expressed in the following way: for any L included in T^*, let us define $Adh(L)$ to be the set $\{u \in T^\omega / LF(u)$ included in $LF(L)\}$. Then the fact that every left factor of an admissible infinite computation is an initial computation is also expressed by

(iii') B^{Inf} is included in $Adh(B^{\text{Init}})$.

In case we have equality between B^{Inf} and $Adh(B^{\text{Init}})$ we shall say that B^{Inf} is closed. This turns out to be a very interesting property which allows reasoning by continuity, or induction, to deduce properties of admissible infinite computations from properties of their finite prefixes. It is well known that it is not true in general: as an example, let us consider example 3 again.

EXAMPLE 3 (continued). For every n, the computation *mutexbegin* followed by n times the action a is a prefix of an admissible infinite computation, but *mutexbegin* followed by ω times a is not admissible, therefore the set B^{Inf} is not closed. ∎

Moreover, if $B^{\text{Init}} = B^{\text{Fin}}$, then it does not matter to set F equal to Q, and if $B^{\text{Inf}} = Adh(B^{\text{Init}})$, then R can be set equal to Q.

2. Properties of transition systems

In this section we assume that a transition system is a pair (Q, T) together with three subsets, I, F, R of Q which define three sets of computations as explained above.

2.1. State mappings. We define the following mappings from $\mathcal{P}(Q)$ into itself: *Reach*, *Coreach*, *Live*, *Loop* by, for P included in Q,

– $q \in Reach(P)$ iff there exists a path (or computation) starting in P and ending in q.

– $q \in Coreach(P)$ iff there exists a path starting in a and ending in P.

– $q \in Live(P)$ iff there exists an infinite path starting in q and going infinitely often through P.

– $q \in Loop(P)$ iff there exists a nonempty path starting and ending in q and going at least once through P.

2.2. Properties of states

It is clear from these definitions that

$$Live(P) = Live(Live(P)) = Coreach(Live(P)),$$

and, if Q is finite,

$$Live(P) = Coreach(Loop(P)).$$

The fact that every initial computation can be extended into an admissible infinite one, i.e., $B^{Init} = LF(B^{Inf})$, is then expressed by: Q included in $Live(R)$. Similarly, the fact that an initial computation can be extended into a terminal one, i.e., $B^{Init} = LF(B^{Fin})$, is expressed by Q included in $Coreach(F)$. A state q is said to be *safe* if every initial computation ending in q can be extended into a terminal computation or into an admissible infinite one. Hence the set of safe states is

$$Safe = Coreach(F) \cap Live(R).$$

The property of being a safe state is not exactly the same as the following one: a state q is said to be *nondeadlocking* if either it is terminal or there exists a transition (q, a, q'), where q' is also a nondeadlocking state. It is not difficult to prove that the set of nondeadlocking states is exactly

$$Ndl = Coreach(F) \cap Live(Q),$$

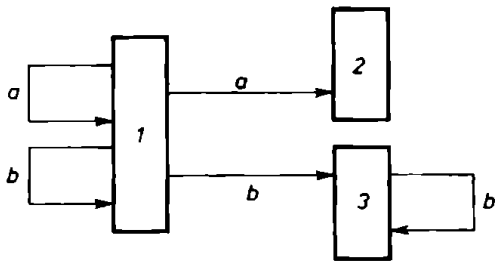
and the difference between the two notions makes no sense in case B^{Inf} is closed, since in this case R can be taken equal to Q.

If we consider the sub-transition system obtained by restricting it to the subset *Safe* (resp. *Ndl*) of Q, then it is a ts whose all states are safe (resp. nondeadlocking) and it is the greatest one having this property.

2.3. Traces. Another point of view about ts is to consider traces of computations instead of computations themselves. The trace of a computation is the sequence of actions appearing in this computation. More precisely the trace of the computation $t_1, t_2, t_3, \ldots, t_n, \ldots$, where $t_i = (q_i, a_i,$

q_i'), is the sequence $a_1, a_2, a_3, \ldots, a_n, \ldots \in A^\infty$. Thus we get three subsets L^{Init} and L^{Fin} of A^*, and L^{Inf} of A^ω. Other properties of ts can be expressed in terms of these trace languages as it is done in [1]. The mapping from T^∞ into A^∞ which associates a trace with a computation is both length preserving and left factor preserving, thus properties such as $B^{\text{Init}} = LF(B^{\text{Fin}})$ are also true for traces. But, unless this mapping is one to one, as it is the case for deterministic ts, the converse does not hold, as shown by the following example.

EXAMPLE 4. Let us consider the following ts:



with $I = \{1\}$, $F = \{2\}$, $R = \{3\}$, and $t_a = (1, a, 1)$, $t_b = (1, b, 1)$, $t_a' = (1, a, 2)$, $t_b' = (1, b, 3)$, $t_b'' = (3, b, 3)$.

Then $B^{\text{Fin}} = \{t_a, t_b\}^* t_a'$; $B^{\text{Inf}} = \{t_a, t_b\}^* t_b' \{t_b''\}^\omega$ and $B^{\text{Init}} = LF(B^{\text{Fin}} \cup B^{\text{Inf}})$. But $B^{\text{Init}} = LF(B^{\text{Fin}})$ and $B^{\text{Init}} = LF(B^{\text{Inf}})$. On the other hand, $L^{\text{Init}} = \{a, b\}^*$, $L^{\text{Fin}} = \{a, b\}^* a$, $L^{\text{Inf}} = \{a, b\}^* b^\omega$. Hence

$$L^{\text{Init}} = LF(L^{\text{Fin}}) = LF(L^{\text{Inf}}). \tag{E}$$

For this ts, the sets of traces satisfy (E) and the sets of computations do not satisfy the similar property:

$$B^{\text{Init}} = LF(B^{\text{Fin}}) = LF(B^{\text{Inf}}). \tag{E'}$$

Moreover, it is impossible to get a ts having the same sets of traces and such that the sets of computations satisfy (E'). ∎

3. Synchronization of transition systems

A system of concurrent processes can be represented by transition systems, and it remains to describe the interactions between the processes of the system. There is a lot of different kinds of such interactions: common access to shared variables, FIFO queues of messages, rendez-vous, message exchange... and various languages for specifying and programming processes use one or several of these different techniques. Our claim is that, at least on the theoretical level, all of them are particular cases of one and the same

general synchronization mechanism, provided that some "objects" involved in these interactions can be also described by transition systems. In Example 2 above, we have shown how a boolean variable could be described this way.

3.1. Synchronization vectors. Let us consider n transition systems $\mathscr{A}_i = (Q_i, T_i)$ over the alphabets A_i, $i = 1, \ldots, n$. We assume that when they run, it is possible to define a sequence of time intervals such that in each interval, each process perform one and only one action. Thus it make sense to speak about global actions performed by this systems of ts: they are constituted of the vector of actions performed by each process in the same interval and thus they are elements of the cartesian product $A_1 \times A_2 \times \ldots \times A_n$. When no interaction exists between the processes, every global action in this cartesian product may be performed by the whole system. When there are interactions, some of them may be forbidden or impossible. Hence specifying the interactions amounts to allowing only a subset of this product as global actions. Moreover, we assume that this subset is constant when the system is running and does not depend on the state of some processes in the system.

This is right for systems which can be called *synchronous* in the sense that in any interval every process performs one action but it is also possible to consider asynchronous systems where at some intervals some processes may stay idle. Then, representing the null action by ε, i.e., a process performing ε remains in the same state, a global action is, in this case, an element of $A_1^0 \times A_2^0 \times \ldots \times A_n^0$, where $A^0 = A \cup \{\varepsilon\}$.

Thus we define a so-called synchronization condition over a system $\mathscr{A}_i = (Q_i, T_i)$ of ts over the alphabet A_i as a subset S of the cartesian product $A_1^0 \times A_2^0 \times \ldots \times A_n^0 - \{\langle \varepsilon, \varepsilon, \ldots, \varepsilon \rangle\}$. Every kind of synchronization between processes can be expressed this way. We shall see below an example of mutual exclusion with a boolean semaphore, where the synchronization condition formalizes the fact that an action performed by a process which modifies the state of the semaphore must be executed in the same time as the semaphore modifies its own state. This kind of synchronization constraint also applies to systems with shared variables (Example 2 above shows how a variable can be represented by a ts): when an action of a process consists in testing or writing a shared variable, this variable must execute the corresponding action. When processes are communicating by bounded or unbounded FIFO queues, then these queues are ts whose states are their own content and actions consist in enqueuing and dequeuing values, and the action of a process consisting in putting a value in the queue or getting a value from the queue is simultaneous with the action of enqueuing or dequeuing. Direct communications between processes in CSP-like languages are described by the fact that the action, executed by a sender, of sending a message to a receiver is simultaneous with the action, executed by the receiver, of receiving the message from the sender. COSY-like interactions

[8] are described by vectors in the form $a = (a_1, a_2, \ldots, a_n)$, where $a_i = a$, if $a \in A_i$, $a_i = \varepsilon$, otherwise.

3.2. Synchronized systems. Now we can explain how such a synchronization condition allows to express the interactions between the component of the system by defining a transition system which represents the synchronized system.

Let $\mathscr{B} = (\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_n; S)$ be the system of ts synchronized by the condition S. The transition system associated with it is the ts over the alphabet S defined in the following way:

- its set of states Q is $Q_1 \times Q_2 \times \ldots \times Q_n$;
- its set of transitions T is defined by

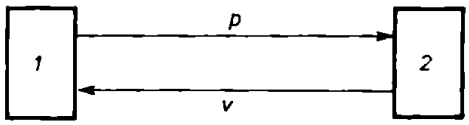
$$((q_1, q_2, \ldots, q_n), (b_1, b_2, \ldots, b_n), (q'_1, q'_2, \ldots, q'_n)) \in T$$

iff

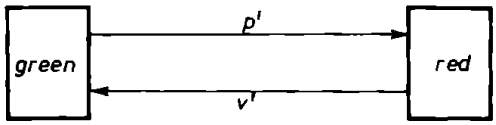
$$(b_1, b_2, \ldots, b_n) \in S \quad \text{and} \quad (q_i, b_i, q'_i) \in T_i, \quad \text{if} \quad b_i \neq \varepsilon,$$

$$q_i = q'_i, \quad \text{otherwise.}$$

EXAMPLE 5. Let us consider the following simplification of Example 1. Here are two identical ts $\mathscr{A}_1$ and $\mathscr{A}_2$ over the alphabet $\{p, v\}$.



Here p and v play the role of *mutexbegin* and *mutexend* of Example 1, and states *1* and *2* the role of *NCS* and *CS*. Now mutual exclusion is performed by the mean of a semaphore $\mathscr{A}_3$ over $\{p', v'\}$;

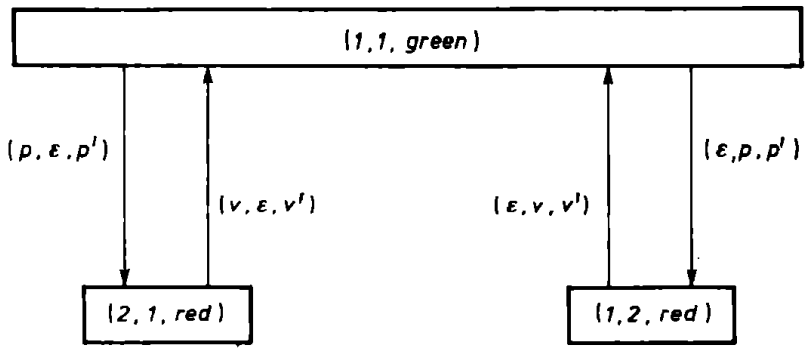


and the synchronization constraint is given by the following set S of vectors:

$$(\varepsilon, p, p'), (p, \varepsilon, p'), (\varepsilon, v, v'), (v, \varepsilon, v')$$

which means that a process can enter its critical section by p, iff, simultaneously, the semaphore goes from *green* to *red* by p', and it can leave it by v, iff the semaphore, simultaneously, goes from *red* to *green* by v'.

The ts defined by this system is:



3.3. Synchronized computations. Let $\mathscr{B} = (\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_n; S)$ be the transition system defined as above and let us consider the projection π_i from T into T_i^0, obviously extended to T^* and T^∞. We define the three sets B^{Init}, B^{Fin}, and B^{Inf} associated with $\mathscr{B}$ in the following way:

$$B^{\text{Init}} = \{u \in T^* / \forall i\, \pi_i(u) \in B_i^{\text{Init}}\},$$

$$B^{\text{Fin}} = \{u \in T^* / \forall i\, \pi_i(u) \in B_i^{\text{Fin}}\},$$

$$B^{\text{Inf}} = \{u \in T^\omega / \forall i\, \pi_i(u) \in B_i^{\text{Inf}} \cup B_i^{\text{Fin}}\}.$$

Notice that the image under π_i of a transition of T can be an empty transition, hence the image of an infinite path can be a finite word. Thus an infinite path in $\mathscr{B}$ is admissible if its infinite projections are admissible and its finite projections are terminated. In other words, in an admissible infinite computation of the system, a component can have a finite computation only if this computation is terminal, which implies that if a component stays forever in the same state, this state has to be a terminal one.

With these definitions it is clear that the initial states of $\mathscr{B}$ are the vectors of initial states, and the terminal states of $\mathscr{B}$ are the vectors of terminal states. But in general it is not true that admissible infinite computations can be characterized by a set R of states of $\mathscr{B}$. This is possible only if $\mathscr{B}$ is given a more intricated definition as in [1]. This suggest the need for an alternative definitions of admissible infinite computations in between the definition in extenso of B^{Inf}, which is too much general, and the definition by repeated states, which is too restrictive.

3.4. Synchrony and asynchrony. Synchronous systems are those in which the vectors of the synchronisation condition do not contain the empty action ε. Thus they can be considered as special cases of asynchronous systems. On the other hand, the ability for a process to stay in a same state can be explicitely expressed by adding an empty transition (q, ε, q) for every state q. Thus asynchronous systems can be seen as special cases of synchronous systems. Indeed, we believe that this second point of view is more interesting

for two reasons: it is possible to admit empty transitions for only certain states and then to mix synchrony and asynchrony, and it forces to explicitely consider arbitrarily long sequences of empty transitions, which is useful when dealing with admissible infinite computations.

4. Analysis of transition systems

In Part 2, we have shown that properties of states could be described by using some functions from $\mathscr{P}(Q)$ into itself. It is possible to consider other such functions, for example functions associated with the operators of some branching time temporal logic [4]. It was already noticed [10] that some temporal logic operators could be defined as fixed points of equations. Also, the operators *Reach* and *Coreach*, defined in Part 2 are solutions of fixed point equations: for P included in Q, we have $X = Coreach(P)$ iff X the least fixed point of the equation

$$X = P \cup Pred(X),$$

where $Pred(X)$ is the set of states q such that there exists a transition (q, a, q') with $q' \in X$.

Similarly $Reach(P)$ is the least fixed point of the equation

$$X = P \cup Succ(X),$$

where $Succ(X)$ is the set of states q' such that $(q, a, q') \in T$ and $q \in X$.

It is also possible to characterize the logical operators introduced in [6] as solutions of fixed point equations.

More generally, it is proposed in [5] to use, as a language for describing sets of states, or, equivalently, properties of states, the set of terms build up with any function which is the solution of a fixed point equation (more precisely: the component of the solution of a system of fixed point equations).

EXAMPLE 6. The set of states reachable from P by a path of any length is $Reach(P)$. The set of states reachable from P by a path of even length is the first component of the least solution of the systems of equations:

$$Even = P \cup Succ(Odd), \quad Odd = Succ(Even). \ \blacksquare$$

It remains to make clear which kind of equations can be used. They are in the form

$$X_i = t_i, \quad i = 1, \ldots, k,$$

where t_i are terms build up from variables X_j, intended to represent sets of

states, set-theoretical operations, and the two functions *Succ* and *Pred* from $\mathscr{P}(Q)$ into itself. Because of the use of the complement, some syntactic conditions on terms are necessary to insure monotonicity of the function represented by a term, and $\mathscr{P}(Q)$ is ordered either by inclusion or by containment; for each variable occurring in a system, one of these two orders is chosen, and in some sense it allows to mix least and greatest fixed points.

EXAMPLE 7. In [2], two different "until" operators are defined. Both are "least" fixed points of the equation

$$X = P \cup (P' \cap Pred(X)).$$

But, for one, X ranges over $\mathscr{P}(Q)$ ordered by inclusion, and for the other, X ranges over $\mathscr{P}(Q)$ ordered the other way. ■

It is not difficult to consider also equation where some variables range over sets of transitions instead of sets of states. It is even possible to consider equations where both kinds of variables occur, provided there exist functions from states to transitions and from transitions to states. Fortunately such functions do exist. They are

$$source \text{ and } target \text{ from } T \text{ into } Q$$

which associate with the transition $t = (q, a, q')$ the state q and the state q'. These functions are additively extended into

$$Source \text{ and } Target \text{ from } \mathscr{P}(T) \text{ into } \mathscr{P}(Q)$$

and their reciprocals are

$$Source^{-1} \text{ and } Target^{-1} \text{ from } \mathscr{P}(Q) \text{ into } \mathscr{P}(T).$$

These functions allow to define *Succ* and *Pred*:

$$Succ(P) = Target(Source^{-1}(P)), \qquad Pred(P) = Source(Target^{-1}(P)).$$

Using these generalized systems of equations, it is possible to define a wider family of sets of states than the one which can be defined by using only *Reach* and *Coreach*, or by using classical temporal operators. However, it can be proved that the function *Loop* cannot be characterized this way.

From an algorithmic point of view, it is possible to compute the solutions of these equations in a time quadratic in the size of the system of equations and quadratic in the size of the transition system. This complexity can be improved in some cases and made linear in the size of the transition system, as it is the case for the algorithms proposed in [4] (cf. note below). Moreover, although it is not expressible as the solution of a system of fixed point equations *Loop* is also computable by an algorithm which is linear in the size of the transition system [11].

Conclusion

Transition systems are known to be an adequate formalism to describe concurrent processes. They are also an adequate mathematical model in the sense that problems about concurrent processes can be precisely stated in this formalism. The theory of transition systems is not yet developped enough to allow to solve these problems but it seems quite evident that any progress in this direction will be of some help in the study of concurrent processes.

Note added in proof. It has been shown in [12] that any system of equations can be solved in time linear in the size of the transition system.

References

[1] A. Arnold, M. Nivat, *Comportements de processus*, In Colloque AFCET "Les mathématiques de l'Informatique" (1982), 35–68.

[2] M. C. Browne, E. M. Clarke, *Automatic circuit verification using temporal logic: two new examples*, Research report, Carnegie-Mellon University (1985).

[3] J. R. Buchi, *On a decision method in restricted second order arithmetic*, In Proc. Int. Congress Logic, Stanford Univ. Press (1962), 1-11.

[4] E. M. Clarke, E. A. Emerson, A. P. Sistla, *Automatic verification of finite state concurrent processes using temporal logic specifications: a practical approach*, Tenth Annual Symp. on Principles of Programming Languages (1983).

[5] A. Dicky, *An algebraic and algorithmic method of analysing transition systems*, Theor. Comput. Sci. 46 (1986), 285–303.

[6] Z. Habasinski, *Process logics: two decidability results*, MFCS'84, Lect. Notes Comput. Sci. 176 (1984), 282–290.

[7] R. M. Keller, *Formal verification of parallel programs*, Comm. A.C.M. 19 (1976), 371–384.

[8] P. E. Lauer, P. R. Torrigiani, M. W. Shields, *COSY, a system specification language based on paths and processes*, Acta Informatica 12 (1979), 109–158.

[9] R. Milner, *A calculus of communicating systems*, Lect. Notes Comput. Sci. 92 (1980).

[10] J. P. Queille, J. Sifakis, *Fairness and related properties in transition systems: a temporal logic to deal with fairness*, Acta Informatica 19 (1983), 195–220.

[11] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput. 1 (1972), 146–160.

[12] A. Arnold, P. Crubillé, *A linear algorithm to solve fixed point equations on graphs*, Rapport I 8632, Univ. Bordeaux I.

Presented to the semester
Mathematical Problems in Computation Theory
September 16–December 14, 1985