Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

A temporal logic approach to the semantics of concurrent systems

Alessandro Fantechi, Stefania Gnesi,
Cosimo Laneve, Gioia Ristori

Nota Interna B4-40
Agosto 1989

A temporal logic approach to the semantics of concurrent systems

A. Fantechi*, S. Gnesi*, C. Laneve**, G. Ristori **

** Istituto di Elaborazione dell'Informazione - C.N.R. - Pisa - Italy*
*** Dipartimento di Informatica - Università di Pisa - Italy*

Abstract

Some results obtained in our research on temporal semantics of languages for the formal description of concurrent systems are presented in this paper.

A compositional linear temporal semantics to a CCS-like language, sound and complete with respect to the operational semantics modulo string equivalence of this language, is given. Then we consider the possibility to enrich the expressive power of this temporal semantics, on one hand in order to express other language operators, on the other hand to obtain the consistency with respect to the observational semantics of the language.

It is shown that the temporal semantics given is a suitable tool for a study of the properties of infinite computations and for an analysis of possible semantics of recursion.

1. Introduction

Several languages for the formal description of concurrent (or more generally, "reactive") systems have been developed; a particular class of such languages is that originated by the Calculus of Communicating Systems defined by Milner [Mil 80], which includes the languages SCCS [Mil 83], TCSP [Bro 84], LOTOS [ISO 88], etc. The languages of this class are based on the description of the behaviour of a process (or of a system) by means of "behaviour" expressions composed by operators that describe the sequential execution of actions, the parallel composition, the synchronization, the nondeterminism, etc., in a manner different for each defined language. We will call a general element of this class an"action prefix" language, but other general names have been used, such as "process description languages" or "process calculi".

A large amount of work has been done to define the notion of equivalence between terms of the same language, to be able to say that a system behave as another under some reasonable assumption.

A specification written in an action prefix language appears to be quite detailed and close to an implementation. Sometimes however a more abstract description of a system - describing its 'abstract' properties instead of the events it can perform - is desired. This can be useful, for example, for relating two specifications by studying their common properties instead of their equivalence.

Temporal logic [Lam 83, Man 81], a particular modal logic which can be used to describe the occurrence of events in time, has been long recognised as adequate to express abstract properties of a concurrent system. This type of logic is needed in order to express properties that relate events occurring in different moments of time (termination, accessibility, etc.) and which cannot be expressed as easily by usual logics (propositional calculus, first order logic).

In this paper we discuss how to provide a concurrent system with a "temporal semantics", i.e. the temporal logic formula expressing the properties of its sequences of execution [Pnu 81].

The advantages of this type of approach when providing the semantics for concurrent systems can be summed up in the following observations:

1) verifying that a system satisfies a given property (expressed by a temporal logic formula) is reduced to verifying that the formula expressing the temporal meaning of the program logically implies the given property formula.

2) verifying the equivalence of programs is reduced to verifying logic equivalence.

This makes it possible to unify these two verification problems, as they amount to verifying the validity of a logic formula.

Our work in this area aims to produce a complete account on the relationships between behavioural specifications using action prefix languages and logic specifications using temporal logic.

This paper illustrates the results and the future directions of our research by starting with giving a compositional linear temporal semantics to a simple example action prefix language. Then we consider the possibility to enrich the expressive power of the temporal semantics given, on one hand in order to express other language operators, on the other hand to obtain the consistency with respect to the observational semantics of the language. In the end we consider the temporal semantics as a suitable tool to study properties of infinite computations and, more in general, recursive definition of processes (both guarded and unguarded).

2. A simple action prefix language

Let us consider a simple action prefix language, which is very close to CCS, employing a finite set of "actions" Act. The operational semantics of this language is defined in terms of transitions that the behaviours are able to perform. We denote a transition by B-a->B', which means that B can perform the action 'a' evolving in the behaviour B'.

syntax	operational semantics	operator
stop	(cannot perform any transition)	inaction
a;B	a;B -a->B	action prefix
B1□B2	B1-a->B1' *implies* B1□B2 -a->B1' B2-a->B2' *implies* B1□B2 -a->B2'	non-deterministic choice
B1‖B2	B1-a->B1' *and* B2-a->B2' *implies* B1‖B2-a-> B1'‖B2'	synchronization
rec x.B(x)	B(**rec** x.B(x)) -a-> B' *implies* **rec** x.B(x) -a-> B'	recursion

Table 1

Table 1 gives the syntax and the operational semantics of our language by recursively defining the transitions of behaviour expressions. The operators are given in order of increasing precedence.

By this language it is possible to express sequences of transitions, which will be finite if the recursion operator is not used, or infinite if (guarded) recursion is used.

3. The compositional linear temporal semantics of the language

Since we are interested in giving the temporal semantics to this action prefix language and the behaviour expressions in the above language can be modelled by finite or infinite, discrete in time, sequences of transitions, we will define a bounded linear time temporal logic, that is having as models finite or infinite sequences. In this logic predicates specify events, i.e. transitions, and formulae contain usual first order logic and basic temporal operators. The atomic formulae are borrowed from the action set (we will call them "transition predicates"). The operators that we will use are the usual first order logic connectives ***true***, ***false***, $\neg$, $\vee$, $\wedge$, together with the temporal operators O *(next)*, *[] (always)*, $\Diamond$ *(eventually)*, $\mathcal{W}$ *(unless)*, $\mathcal{U}$ *(until)* and a maximal fixed point constructor $\nu\xi.\,\chi\,(\xi)$.

In order to define the semantics of temporal formulae, we need to introduce the concept of a model. A model σ is a sequence, finite or infinite, of actions (corresponding to a particular process execution). In Table 2 we define inductively what we mean by an interpretation $<\sigma,i>$ (a pair constituted by a model, and a discrete instant of time) to satisfy a formula ϕ; this is expressed by the notation: $\sigma,i \models \phi$.

		Propositional calculus operators			
$\sigma,i \models \mathbf{true}$	$= t,$		$\sigma,i \models \mathbf{false}$	$= f.$	
$\sigma,i \models a$	<u>iff</u>	$\sigma(i) = a$	$\sigma,i \models \neg\phi$	<u>iff</u>	<u>not</u> $(\sigma,i \models \phi)$
$\sigma,i \models \phi \vee \psi$	<u>iff</u>	$(\sigma,i \models \phi)$ <u>or</u> $(\sigma,i \models \psi)$	$\sigma,i \models \phi \wedge \psi$	<u>iff</u>	$(\sigma,i \models \phi)$ <u>and</u> $(\sigma,i \models \psi)$
		Linear temporal operators			
$\sigma,i \models O\phi$	<u>iff</u>	$\text{length}(\sigma) \leq i$ <u>or</u> $\sigma,i+1 \models \phi$	$\sigma,i \models \nu\xi.\,\chi\,(\xi)$	<u>iff</u>	<u>for all</u> $k>0$ $\sigma,i \models \chi^k(\mathbf{true})$
$\sigma,i \models []\phi$	<u>iff</u>	for all $j \geq i$: $(\sigma,j \models \phi)$	$\sigma,i \models \Diamond\phi$	<u>iff</u>	<u>exists</u> $j \geq i$: $(\sigma,j \models \phi)$
$\sigma,i \models \phi\,\mathcal{U}\,\psi$	<u>iff</u>	<u>exists</u> $k \geq i$: $(\sigma,k \models \psi)$ <u>and</u> <u>for all</u> j : $i \leq j < k$ <u>and</u> $(\sigma,j \models \phi)$	$\sigma,i \models \phi\,\mathcal{W}\,\psi$	<u>iff</u>	$\sigma,i \models []\phi$ <u>or</u> $\sigma,i \models \phi\,\mathcal{U}\,\psi$

Table 2

The formulae of our logic are interpreted on the sequences of actions performed, using propositions which are considered true if the system is able to perform the corresponding action. The obtained formulae can have an obvious interpretation in models which are sequences of actions.

The formulae of our logic are very close to behavioural specifications, therefore the idea is to characterize all behaviours of a program through one temporal logic formula. We will use the temporal logic presented to provide the linear temporal semantics for our example language, which admits only interleaved executions, that is, in which actions are performed one at a time.

In the following we give the temporal semantics of the example language by defining a function $\mathbb{L}$ which associates a temporal formula to each construct of the language by means of a set of syntax-directed clauses given in a denotational style.

The language constructs are behaviour expressions, and their meaning is taken to be a predicate on possible execution sequences generating from that behaviour expression.

The type of the semantic function is hence: $\mathbb{L} : BE \rightarrow TL$,
where *BE* is the space of behaviour expressions of our language and *TL* is the space of temporal formulae.

In Table 3 the clauses defining the function $\mathbb{L}$ by structural induction are listed. The meaning of the process which cannot perform any action is given by the formula **false**. Action prefix a;B is represented by the formula having as models the sequences beginning with a, followed by the sequences performed by B. The meaning of the non-deterministic choice is given by the disjunction of the meanings of the component behaviours, while the synchronization is represented by the conjunction of the component behaviours. Note that any formulae of the type $a \wedge b$, with $a \neq b$ and $a, b \in Act$, is equated to **false**, as expected by the operational meaning of synchronization.

The semantics of recursion can be given by maximal fixed point, which amounts to consider the maximal set of action sequences that constitute a meaningful semantics for recursion. The obtained semantics denoted by $\mathbb{L}$ is presented in Table 3.

For simplicity, the given semantics implicitly uses the notion of an environment where language variables are associated with logic variables with the same name.

This semantics is compositional, i.e. the meaning of a program is given by a composition of the meanings of its components.

Linear temporal semantics

$\mathbb{L}$(stop) =	**false**	$\mathbb{L}$(a; B) =	$a \wedge O\, \mathbb{L}(B)$
$\mathbb{L}$(B1 [] B2) =	$\mathbb{L}(B1) \vee \mathbb{L}(B2)$	$\mathbb{L}$(B1 || B2) =	$\mathbb{L}(B1) \wedge \mathbb{L}(B2)$
$\mathbb{L}$(rec x.B(x)) =	$\nu x.\, \mathbb{L}(B(x))$	$\mathbb{L}$(x) =	x

Table 3

3.1 Parallel composition as interleaving

One of the most common operators in action prefix languages, which has not been considered in our example language, is the parallel composition operator that specifies the interleaved executions of two behaviours.

It is not possible to give temporal semantics to interleaving in a compositional way as easily as for the other constructs: a possible approach is that introduced in [Bar 84, Bar 85], in which a process P is considered as immersed in an environment constituted by other processes which can perform their own actions. The semantics of P will be given considering that P-actions are interleaved with those of the environment, by using a special proposition '*e*', denoting an unknown environment action. When a process P is composed (by interleaving operators) with another process Q, some of the actions performed by the environment of P will be those performed by Q, and vice versa.

In Table 4 we introduce the syntax and the operational meaning of the interleaving operator and in Table 5 the modified semantics considering this new operator.

syntax	operational semantics	operator
B1|||B2	B1-a->B1' *implies* B1|||B2-a-> B1'|||B2 B2-a->B2' *implies* B1|||B2-a-> B1|||B2'	interleaving

Table 4

The definition of the temporal meaning of the interleaving could confuse actions performed by B1 with the same actions performed by B2. To overcome this problem we could identify (by a substitution) the owner of the actions. The identity of the owner should then be dropped from the formula describing the behaviour of the composed process.

Temporal semantics with external action

$\mathbb{L}(\text{stop}) =$	$e\ \mathcal{W}\ \text{false}$	$\mathbb{L}(a; B) =$	$e\ \mathcal{W}\ (a \wedge \bigcirc\, \mathbb{L}(B))$
$\mathbb{L}(B1 [] B2) =$	$\mathbb{L}(B1) \vee \mathbb{L}(B2)$	$\mathbb{L}(B1 \| B2) =$	$\mathbb{L}(B1) \wedge \mathbb{L}(B2)$
$\mathbb{L}(\text{rec } x.B(x)) =$	$\nu x.\ \mathbb{L}(B(x))$	$\mathbb{L}(x) =$	x

$$\mathbb{L}(B1 ||| B2) = \mathbb{L}(B1)[(e \vee \bigvee_{f \in \alpha(B2)} f)/e] \wedge \mathbb{L}(B2)[(e \vee \bigvee_{f \in \alpha(B1)} f)/e]$$

Table 5

The introduction of the interleaving operator has produced the need of a more complex temporal semantics; in general the addition of language operators increases the complexity, so that in some case the introduction of new logic operators is required: in [Fan 88a] the semantics of Basic LOTOS has been given defining two logic operators which model the relabelling and the disabling operators.

The use of the temporal semantics implies an equivalence relation: two processes have the same meaning if the associated formulae are logically equivalent. From the meaning of temporal logic formulae follows that two equivalent processes will have the same set of sequence models. This fact recalls of the "string equivalence" [Hoa 81] which considers two processes to be equivalent if they are able to perform the same sequences of actions (traces).

The linear temporal semantics of this language, presented here, has been shown to be sound and complete with respect to the operational semantics modulo string equivalence [Fan 88b].

4. A compositional branching temporal semantics of the language

We should recall that usually the operational semantics of action prefix langueges, like CCS, is given by observational equivalence. The fundamental difference between observational and string equivalence is that the equivalence classes are, respectively, classes of action labelled trees (the synchronization trees of [Mil 80]) and classes of sets of sequences of actions. This is due to the difference between the basic model for the process used by the operational semantics, i. e. the labelled transition system, and the sequence model used for linear time temporal logic. This causes the different discriminating power: the observational equivalence has a finer discriminating power with respect to the string one.

In order to re-acquire the discriminating power of the observational equivalence we would use another version of temporal logic (branching time), in which models are trees of actions.

We choose a branching time logic that is a subset of CTL* [Eme 86], with the addition of a maximal fixed point constructor, to give the branching temporal semantics to our language .

The formulae in this logic are interpreted on the states of a model structure $\mathcal{M}$ which can be seen as a synchronization tree. The temporal operators of this logic, namely *AX (for all next states)*, *EX (exists a next state)* are interpreted as shown in Table 6. The classical propositional operators and the fixed point constructor are defined analogously to Table 2.

	Propositianal calculus operators	
$s \models_{\mathcal{M}} a$	<u>iff</u>	<u>exists</u> $s' \in \mathcal{M}$: s' -a-> s
	Branching temporal operators	
$s \models_{\mathcal{M}} AX\,\phi$	<u>iff</u>	<u>for all</u> $s' \in \mathcal{M}$, s -a-> s' : $s' \models_{\mathcal{M}} \phi$
$s \models_{\mathcal{M}} EX\,\phi$	<u>iff</u>	<u>exists</u> $s' \in \mathcal{M}$, s -a-> s' : $s' \models_{\mathcal{M}} \phi$

Table 6

The branching temporal semantics of the example language can be given similarly to the linear one and it is shown in Table 7.

Branching temporal semantics

$\mathcal{T}(B)$	=	$\neg$EX **true** $\vee$ AX $\mathcal{B}(B)$		
$\mathcal{B}$(**stop**)	=	**false**	$\mathcal{B}(a; B)$ =	$a \wedge \mathcal{T}(B)$
$\mathcal{B}(B1 \,\square\, B2)$	=	$\mathcal{B}(B1) \vee \mathcal{B}(B2)$	$\mathcal{B}(B1 \| B2)$ =	$\mathcal{B}(B1) \wedge \mathcal{B}(B2)$
$\mathcal{B}$(rec x.B(x))	=	$\nu x.\ \mathcal{B}(B(x))$	$\mathcal{B}(x)$ =	x

Table 7

However, it can be seen that this semantics is isomorphic to the linear one and hence sound and complete with respect to the operational semantics modulo string equivalence; therefore it is not satisfactory in order to express the observational equivalence. Actually, a compositional branching temporal semantics for our language, sound and complete with respect to the operational semantics modulo bisimulation, can be obtained by a more complex definition, for example defining new logic operators more suitable to model the synchronization [Ris 89].

A different approach is to start directly by defining an expressive logic for the language. This has been followed in [Hen 85], where an expressive logic (with respect to the observational equivalence, but not to properties of infinite computations) is defined to give logic semantics, non compositional, to CCS programs.

5. Justice and fairness properties

We have given previously some relationships between the temporal semantics of the language and its operational semantics. However, the main interest in giving temporal semantics is to be able to express properties which cannot be expressed by the operational semantics. Interesting cases are the fairness and justice properties. Both properties refer to infinite computations: a process is fair if in performing infinitely often a choice among more alternatives, each alternative will eventually be executed; justice is instead the requirement that if a process is constantly able to choose in a set of actions (that is, the actions of the set are constantly enabled) then eventually one action of the set will be performed [Pnu 88].

The reason for which these properties are not expressible with the operational semantics is in the model which underlies it. Labelled transition systems cannot distinguish fair or unfair behaviours, or behaviours with or without justice since such systems are not sensitive to computations which may be different at infinite. For this reason Fair Transition Systems have been introduced by Pnueli, as a model for fair concurrent systems [Pnu 88].

It is instead well known that temporal logic can express fairness and justice requirements; in Table 3 two temporal formulae expressing the fairness and justice constraints as defined above are presented.

Fairness (Strong Fairness)

$$[]\Diamond (\bigvee_{a \in S} a) \Rightarrow (\bigwedge_{a \in S} \Diamond a)$$

where $S \subseteq Act$

Justice (Weak Fairness)

$$[] (\bigvee_{a \in S} a) \Rightarrow (\bigvee_{a \in S} []\Diamond a)$$

Table 8

In general action prefix languages have no syntax nor semantics to express that a behaviour is, for example, fair and another is not; it is hence impossible to verify any similar property on specifications expressed in such languages. For this purpose it is necessary to augment the language by syntactic extensions or by semantic enhancements.

The first approach - the syntactic extension - is proposed for CCS in [Par 85], where the behaviour syntax is augmented with temporal logic formulae which allow to express such properties.

Another approach is to enhance the semantics of the actions of the language. A way to define a justice semantics for our action prefix languages can be to assume that the external environment "properly" favours the execution of a process. By "properly" we can intend that the environment is always ready to offer/accept an action that the process is able to do; this guarantees justice. We can consider this assumption, by modifying the semantics of the language in Table 5 so that non-environment actions must eventually occurr, modifying only the meaning of the action prefix operators: $\mathcal{L} (a; B) = e \, \mathcal{U} (a \wedge O \, \mathcal{L} (B))$.

This shows that the method chosen to give the temporal semantics to our language is flexible enough to express particular assumptions on the semantics of the language, which allow to verify certain classes of properties.

6. Guarded and unguarded recursion

One of the flaws in the definition of an action prefix languages like CCS [Mil 80] is related to the possibility of syntactically writing expressions which have an undefined behaviour, by means of so called "unguarded recursion".

The concept of guarded and unguarded recursion can be defined as follows [Mil 86]:

1. A free occurrence of x in an expression E is said *guarded* in E if it occurs within some

subexpression a;F of E. Otherwise it is said *unguarded* in E.

2. An expression of the form: **rec** x.E where x is guarded in E, is said a *guarded recursion.* Conversely , it is said *unguarded recursion* if x is unguarded in E.

This possibility was ruled out in the original work on CCS by means of a sort of static semantic checks. Most work on the analysis and development of action prefix languages have inherited this assumption of "well-guarded-ness", and so this problem is not extensively treated in the literature.

In our research on the application of temporal logic to concurrent systems, we have noticed that temporal semantics, that is the temporal logic formula expressing the properties of a program , can be a proper tool for analyzing the issue of recursion.

Previous works on the logic treatment of the semantics of action prefix languages [Bar 85, Bar 86, Hen 85] have not covered the issue of unguarded recursion.

In [Fan 89] we show that minimal and maximal fixed point temporal semantics can be given for this language. The minimal semantics can be obtained, in reference to Table 3, by substituting the minimal fixed point operator to the maximal one.

The minimal fixed point constructor can be defined, for duality vs. the maximal one, as following:

$$\sigma,i \models \mu\xi.\,\chi(\xi) \qquad \underline{\text{iff}} \qquad \underline{\text{exists}}\ k>0 \quad \sigma,i \models \chi^k\,(\textbf{false}).$$

The differences between the minimal and maximal interpretations of guarded and unguarded recursion in our language are analyzed, giving the result that they differ, for guarded recursive processes, only for infinite behaviours while they are the same for finite models: For unguarded recursive processes they differ completely, and actually we obtain some interesting results that refine the interpretation given by the operational semantics.

As an example, the simplest case of unguarded recursion, **rec** x.x, is equated to **stop** by the minimal semantics; the maximal interpretation is, on the contrary, the process which can behave as any other process.

These results can be generalized to other, more complex, action prefix languages; the main point in this direction is to study the classes of language operators which maintain the given properties of minimal and maximal interpretation of recursion.

7. Conclusions

In this paper we have presented a survey of some results obtained in our research on temporal semantics.

A conclusion we can draw from these results is that temporal semantics is able to provide, possibly by modifying the logic, different semantics to the same language, exploiting different expressive power with respect to equivalences or to property definitions (in particular to properties of infinite sequences). It would be useful to be able to derive different semantics from a single general framework, mantaining a notion of mutual consistency between logic and operational semantics;

interesting works in this respect are [Abr 88, Win 88].

On the other hand, any practical application of the proposed approach to real system specifications needs automatic supports. The verification that a behavioural specification satisfies a given temporal logic formula expressing a desired property can be achieved: i) by using a model checker [Cla 86], ii) by building a proof system directly derived from the temporal semantics [Bar 86] or iii) by reducing the problem to the verification of the validity of a formula, once given the temporal semantics of the specification. In its turn, the latter verification can be achieved by means of a decision procedure [Gou 87] for the logic or by theorem proving on the set of axioms of the logic. Our research will investigate the automatization of these possibilities.

References

[Abr 88] S. Abramsky, "Domain Theory in Logical Form", Imperial College Research Report DOC 88/15, September 1988.

[Bar 84] H. Barringer, R. Kuiper, A.Pnueli: "Now you may Compose Temporal Logic Specifications", Proc. 16th ACM Symposium on the Theory of Computing, 1984.

[Bar 85] H. Barringer, R. Kuiper, A.Pnueli: "A Compositional Temporal Approach to a CSP-like Language", in E.J. Neuhold and G. Chroust eds., Formal Models of Programming, IFIP, North-Holland, pp. 207-227.

[Bar 86] H. Barringer, "Using Temporal Logic in the Compositional Specifications of Concurrent Systems", University of Manchester, Technical Report UMCS-86-10-1, October 1986.

[Bro 84] S.D. Brookes, C.A.R. Hoare, A.D.Roscoe, "A Theory of Communicating Sequential processes", Journal of ACM, vol. 31, n. 3, pp.560-599, 1984.

[Cla 86] E. M. Clarke, E.A. Emerson, A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification", ACM Transactions on Programming Languages and Systems, vol. 8, n. 2, pp. 244-263, 1986.

[Eme 86] E.A. Emerson, J.Y. Halpern, ""Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic", Journal of the ACM, vol. 33, n. 1, Jan. 1986, pp. 151-178.

[Fan 88a] A. Fantechi, S. Gnesi, C. Laneve, "Principles for a Temporal Semantics of LOTOS", I.E.I. Internal Report B4-36, August 1988.

[Fan 88b] A. Fantechi, S. Gnesi, C. Laneve, "A Proof of Simple Abstractness for the Temporal Semantics of LOTOS", I.E.I. Internal Report B4-38, August 1988.

[Fan 89] A. Fantechi, S. Gnesi, C. Laneve, "A Logic Approach to Guarded and Unguarded Recursion", I.E.I. Internal Report B4-06, March 1989.

[Gou 87] G.D. Gough, H. Barringer, "A Semantics Driven Temporal Verification System", University of Manchester, Technical Report UMCS-87-12-5, December 1987.

[Hen 85] M. Hennessy, R. Milner, "Algebraic Laws for Nondeterminism and Concurrency", Journal of ACM, vol. 32, n. 1, pp.137-161, 1985.

[Hoa 81] C.A.R. Hoare, "A Model for Communicating Sequential Processes", Technical Monograph Prg-22, Computing Laboratory, University of Oxford, 1981.

[ISO 88] ISO - Information Processing Systems - Open System Interconnection, LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, DIS 8807, 1987.

[Lam 83] L. Lamport, "What Good is Temporal Logic?", Proceedings of IFIP'83, North Holland, 1983, pp.657-668.

[Man 81] Z. Manna, A. Pnueli, "Verification of Concurrent Programs: The Temporal Framework", in R.S. Boyer, J.S. Moore, eds., "Correctness Problem in Computer Science", Academic Press, 1981, pp.215-273.

[Mil 80] R. Milner : "A Calculus of Communicating Systems", Lecture Notes in Computer Science vol. 92, 1980.

[Mil 83] R. Milner, "Calculi for Synchrony and Asynchrony", Theoretical Computer Science, vol. 25, 1983, pp. 267-310.

[Mil 86] R. Milner, "A Complete Axiomatisation for Observational Congruence of Finite-state Behaviours", University of Edinburgh, LFCS Report Series, ECS-LFCS-86-8, August 1986.

[Par 85] J. Parrow, R. Gustavsson, "Modelling Distributed Systems in an Extension of CCS with Infinite Experiments and Temporal Logic", in Y. Yemini, R. Strom, S. Yemini, eds., "Protocol Specification, Testing, and Verification, IV", North-Holland, 1985, pp. 309-348.

[Pnu 81] A. Pnueli, "The Temporal Semantics of Concurrent Programs", Theoretical Computer Science, vol.13, 1981, pp.45-60.

[Pnu 88] A. Pnueli, "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends", REX School /Workshop, Noordwijkerhout, May 1988.

[Ris 89] G. Ristori, "Semantica Temporale 'Branching' e Linguaggi Basati su Sistemi di Transizione", Tesi di Laurea, Università di Pisa, in preparazione.

[Win 88] G. Winskel, "A Category of Labelled Petri Nets and Compositional Proof System", Proceedings 3rd Annual Symposium on Logic in Computer Science, Edinburgh, Scotland, July 1988, pp. 142-154.