J. Symbolic Computation (1996) **11**, 1–000

Deriving Liveness Goals from Temporal Logic Specifications¶

C. CALEIRO†, G. SAAKE‡ AND A. SERNADAS†

†*Dep. Matemática, IST, Av. Rovisco Pais, 1000 Lisboa, Portugal*
‡*Univ. Magdeburg, Universitätsplatz, D-39016 Magdeburg, Germany*

(Received 27 May 1996)

A propositional temporal logic is briefly introduced and its use for reactive systems specification is motivated and illustrated. G-automata are proposed as a new operational semantics domain designed to cope with fairness/liveness properties. G-automata are a class of labelled transition systems with an additional structure of goal achievement which forces the eventual fulfilment of every pending goal. An algorithm is then presented, that takes a finite system specification as input and that, by a stepwise tableaux analysis method, builds up a canonical G-automaton matching the specification. Eventuality formulae correspond to goals of the automaton their satisfaction being thus assured. The direct execution of G-automata, and consequently of specifications, is then discussed and suggested as an alternative approach to the execution of propositional temporal logic. A short overview of the advantages of applying the techniques to the specific field of database monitoring is presented.

Keywords: reactive system, propositional temporal logic, tableau, eventuality, goal, G-automaton, execution, monitoring.

1. Introduction

The use of temporal logic has been widely explored both on the fields of specification and certification of properties of reactive systems (Pnueli, 1977), (Sernadas, 1980), (Fiadeiro and Maibaum, 1992), (Clarke, Grumberg and Kurshan, 1992), (Manna and Pnueli, 1992), (Manna and Pnueli, 1993), (Sernadas, Sernadas and Costa, 1995), (Sernadas, Sernadas and Ramos, 1996) and in monitoring (Hülsmann and Saake, 1991), (Kung, 1984), (Lipeck and Saake, 1987), (Schwiderski, Hartmann and Saake, 1994). The advantages are known to lie on the clear declarative formalization of the system at hand and on the use of temporal verification techniques to prove properties of the specified systems. Temporal logic specification has also given an important contribution towards the establishment of suitable compositional specification frameworks (Barringer, Kuiper and Pnueli, 1984).

¶ This work was partly supported by CEC under ESPRIT-III BRA WG 6071 IS-CORE (Information Systems – COrrectness and REusability), WG 6112 COMPASS (COMPrehensive Algebraic approach to System Specification development) and WG 8319 ModelAge (A Common Formal **Model** of Cooperating Intelligent **Agents**).

0747–7171/90/000000 + 00 $03.00/0

We address the subject of connecting temporal specification with an operational semantic domain build around the notion of labelled transition system. The problem, identified in (Sernadas, Costa and Sernadas, 1994) and first dealt with in (Caleiro, 1993), resides on establishing a canonical operational semantics of temporal specifications, and consequently on finding a notion of labelled transition system which copes well with liveness and fairness issues. Moreover, the results obtained in (Sernadas, Sernadas and Caleiro, 1996) concerning denotational semantics also stressed the need for providing an adequate operational counterpart. For the purpose, and restricting our attention to propositional temporal logic, we present G-automata, which are labelled transition systems with stuttering, additionally equipped with a structure of goals whose intention is to restrict acceptance. These are closely related to assuming fairness assumptions on the execution of transition systems (Manna and Pnueli, 1992), (Manna and Pnueli, 1993) but use an approach somehow similar to the acceptance structure of other, well known, definitions of automata on infinite computations such as those of Büchi, Muller or Streett automata (Thomas, 1990) or variants of these, eg. $\forall$-automata (Manna and Pnueli, 1987). However, G-automata arise, in our opinion, much more naturally, while still emulating their expressive power. Mainly due to the fact that they explore transitions rather than states to formulate the acceptance condition on computations, which seems much more natural since the performance of transitions is the only way a system may evolve in order to satisfy some commitment. Using such a semantic domain we manage to fill in the gap between specification and operational semantics. We provide a constructive method to obtain the operational semantics of a system specification by means of a tableaux-like method based on the decision procedure for propositional temporal logic satisfiability described in (Ben-Ari, Pnueli and Manna, 1983). The approach is therefore similar to the "traditional" methods for synthesizing models from temporal specifications (Clarke and Emerson, 1981), (Manna and Wolper, 1984), (Pnueli and Rosner, 1988). However, we do not just choose a computation meeting the specification as in (Manna and Wolper, 1984) but we manage to build a G-automaton which generates all of its models. Note that fairness or liveness properties can be specified, having an operational counterpart reflected on the acceptance condition of the obtained automaton.

The direct execution of the specifications can therefore be discussed in the context of possible strategies for implementing G-automata features. We therefore expect to give a contribution to the field of executable temporal logics (Fisher and Owens, 1995). Although the logic we present has some useful specific features such as a clear distinction between state variables, action occurrence and action enabling (Sernadas, Sernadas and Costa, 1995), our main concern is not to propose a new temporal programming language but rather suggest a different way of implementing the execution of temporal logic statements based upon the operational model introduced. We compare with related work and point out how the results can be useful, namely in database monitoring applications (Kung, 1984), (Lipeck and Saake, 1987). We end up with an outlook of possible developments of the proposed techniques.

Through the paper, we assume that the reader is conversant with the field of temporal logic specification, for instance at the level of (Emerson, 1990), (Goldblatt, 1987), and with fairness and liveness issues (Francez, 1986), (Gabbay, Pnueli, Shelah and Stavi, 1980).

In section 2 we introduce and illustrate the use of temporal logic in systems specification. The temporal language is defined and several kinds of specifiable properties are mentioned. The fundamental aspect of the logic is its ability to distinguish between occurrence and enabling of an action. Section 3 is devoted to a full description of the class of automata we propose. We pay close attention to the structure of pending goals and corresponding fulfilment associated with each automaton. Parallel composition is studied as an example of the degree of compositionality obtainable on such a domain. We also define the satisfaction relation between G-automata and the formulae of the adopted specification logic. The core section of the paper is the fourth. It is where the method of synthesis of a G-automaton from a given system specification is described. A tableaux calculus for the logic at hand is introduced and used as a way of reasoning about "next state". Section 5 is dedicated to outlining strategies for directly executing specifications via the implementation of the synthesized G-automata. We also compare the techniques with other existing approaches to executing propositional temporal logic specifications. At last, section 6 discusses the results obtained, their connection with previous work and applications to monitoring. It also outlooks possible extensions to this work.

2. Temporal specification of reactive systems

We consider a reactive system to be some entity able to sequentially perform actions. Between action occurrences some local state is reached and may be observed by associating boolean values to it. One of the most important observable properties of a state is its action menu, that is those actions whose occurrence is possible. Therefore, as in (Sernadas, Costa and Sernadas, 1994), we consider a system specification to be a description of its actions and observations as well as of the enabling conditions of actions and of the effects of each occurrence on observations. This corresponds to restricting the possible combinations of action occurrences and observations allowed by the alphabet.

We start by shortly introducing the envisaged specification logic.

2.1. The specification logic

The specification logic we use, a simplified version of OSL (Object Specification Logic) (Sernadas, Sernadas and Costa, 1995), is a propositional multilinear temporal logic. Multilinear in the exact sense that it distinguishes action occurrence from action enabling (Sernadas, Sernadas and Caleiro, 1996). So, if a is some action (or prime action, as we shall name it), $\triangledown a$ stands for "a occurs" and $\diamond a$ for "a is enabled to occur".

Definition 2.1. (Signature)
A *signature* is a pair $\Sigma = (\Sigma_{obs}, \Sigma_{act})$ of sets, to whose elements we call, respectively, observation symbols and prime action symbols. □

Now, we define the envisaged temporal language over a given signature. For the purpose, throughout the remaining of the subsection, we consider fixed a signature Σ.

Definition 2.2. (Atoms)
The set of *atoms* over Σ is $Atom_\Sigma = \{b : b \in \Sigma_{obs}\} \cup \{\triangledown a : a \in \Sigma_{act}\} \cup \{\diamond a : a \in \Sigma_{act}\}$. □

DEFINITION 2.3. (Formulae)
The set $Form_\Sigma$ of *formulae* over Σ is the least set such that:

$Atom_\Sigma \subseteq Form_\Sigma$;
$(\neg\,\varphi), (\varphi \Rightarrow \psi), (\mathsf{X}\,\varphi), (\varphi\,\mathsf{U}\,\psi) \in Form_\Sigma$ if $\varphi, \psi \in Form_\Sigma$. □

As usual, the temporal language can be extended with some useful abbreviations:

$(\varphi \vee \psi) \stackrel{abv}{\equiv} ((\neg\,\varphi) \Rightarrow \psi)$;
$(\varphi \wedge \psi) \stackrel{abv}{\equiv} (\neg((\neg\,\varphi) \vee (\neg\,\psi)))$;
$(\varphi \Leftrightarrow \psi) \stackrel{abv}{\equiv} ((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$;
$(\mathsf{G}\,\varphi) \stackrel{abv}{\equiv} (\varphi\,\mathsf{U}\,(\varphi \wedge (\neg\,\varphi)))$;
$(\mathsf{F}\,\varphi) \stackrel{abv}{\equiv} (\neg(\mathsf{G}(\neg\,\varphi)))$.

We introduced four temporal operators. The "next" operator (X) refers, as usual, to the following instant. The "until" operator (U) we adopt is weak and includes the evaluating point. The derived ones have the usual meanings, namely, "henceforth" for (G) and "eventually" for (F). Both include the evaluating point.

In order to define the satisfaction of formulae we consider *life-structures* as sets of linear *life-cycles*. A life-cycle consists, therefore, in a sequence of sets of atoms, where every occurring prime action must be enabled. To such sets of atoms we call *snapshots*. Consequently, we also require that enabling corresponds to possible execution in an alternative life-cycle of the life-structure.

DEFINITION 2.4. (Snapshots, life-cycles and life-structures)
A *snapshot* over Σ is a set $snp \subseteq Atom_\Sigma$ such that if for some $a \in \Sigma_{act}$, $\triangledown a \in snp$ then also $\diamond a \in snp$. We denote by Snp_Σ the set of all snapshots over Σ. For each $snp \in Snp_\Sigma$ we denote by $snp \setminus \triangledown$ the set $snp \cap (\{b : b \in \Sigma_{obs}\} \cup \{\diamond a : a \in \Sigma_{act}\})$.

A *life-cycle* over Σ is a map $\lambda : \mathbb{N} \rightarrow Snp_\Sigma$. We denote by Λ_Σ the set of all life-cycles over Σ. For each $i \in \mathbb{N}$ and $\lambda \in \Lambda_\Sigma$ we denote by $[\lambda[_i$ the prefix of length i of the sequence λ.

A *life-structure* over Σ is a set $\Lambda \subseteq \Lambda_\Sigma$ such that, for each $\lambda \in \Lambda$, $a \in \Sigma_{act}$ and $i \in \mathbb{N}$, $\diamond a \in \lambda_i$ iff there exists $\lambda' \in \Lambda$ with $]\lambda]_i =]\lambda']_i$, and $\lambda_i \setminus \triangledown = \lambda'_i \setminus \triangledown$ such that $\triangledown a \in \lambda'_i$. □

DEFINITION 2.5. (Satisfaction)
Let λ be a life-cycle over Σ. The *satisfaction* of formulae in $Form_\Sigma$ by λ at each $i \in \mathbb{N}$ is inductively defined as follows:

$\lambda \models_i b$ iff $b \in \lambda_i$;
$\lambda \models_i \diamond a$ iff $\diamond a \in \lambda_i$;
$\lambda \models_i \triangledown a$ iff $\triangledown a \in \lambda_i$;
$\lambda \models_i (\neg\,\varphi)$ iff not $\lambda \models_i \varphi$;
$\lambda \models_i (\varphi \Rightarrow \psi)$ iff $\lambda \models_i \psi$ or not $\lambda \models_i \varphi$;
$\lambda \models_i (\mathsf{X}\,\varphi)$ iff $\lambda \models_{i+1} \varphi$;
$\lambda \models_i (\varphi\,\mathsf{U}\,\psi)$ iff either, for some $j \geq i$, $\lambda \models_j \psi$ and for all $i \leq k < j$, $\lambda \models_k \varphi$ or for every $j \geq i$, $\lambda \models_j \varphi$.

We say that λ satisfies a formula φ over Σ, $\lambda \models \varphi$, iff $\lambda \models_i \varphi$ for every $i \in I\!N$. We also say that λ satisfies a set of formulae $\Phi \subseteq Form_\Sigma$, $\lambda \models \Phi$, iff $\lambda \models \varphi$ for every $\varphi \in \Phi$. And we extend the definition to life-structures. We say a life-structure Λ over Σ satisfies a formula φ, $\Lambda \models \varphi$, iff $\lambda \models \varphi$ for each $\lambda \in \Lambda$. Analogously, Λ satisfies a set of formulae Φ, $\Lambda \models \Phi$, iff $\Lambda \models \varphi$ for each $\varphi \in \Phi$. □

Obviously, a life-structure can be seen as a set of life-trees, life-cycles corresponding to infinite paths in a tree. Enablings can therefore be derived, at each node of a tree, by looking for possible occurrences in alternative nodes.

The temporal logic we use is non-anchored (Manna and Pnueli, 1989) and is not equipped with a birth atom whose satisfaction is only held at 0. This is because we are interested in specifying overall properties of systems behaviour with no particular interest in initial constraints. Note, however, that adding it would bring no relevant technical problems throughout the remainder of the paper.

Definition 2.6. (Specification and satisfaction)
A *specification* is a pair $spec = (\Sigma, \Gamma)$, where Σ is a signature and $\Gamma \subseteq Form_\Sigma$ is a finite set of formulae (usually referred to as axioms). A life-structure Λ over Σ satisfies $spec$, $\Lambda \models spec$, iff $\lambda \models \Gamma$. □

2.2. Specification examples

It must be clear that, due to the definition of snapshot, formulae of the form $(\triangledown a \Rightarrow \diamond a)$ are always satisfied. It is also intended, as made clear by the definition of life-structure, that the meaning of $\diamond a$ is exactly that a can occur (i.e., enabling of a prime action is an observation whose value at each instant reflects its possible occurrence immediately after). We also require that the systems are open towards the environment. By this we mean that occurrences may be finitely delayed, in any life-cycle, in order to allow interleaving with life-cycles of other systems (Sernadas, Sernadas and Caleiro, 1996). This shall mean that, in general, a formula like $\triangledown a$ cannot be expected to hold.

We start by taking a brief look at some classes of formulae commonly used in systems specification (Manna and Pnueli, 1989), (Manna and Pnueli, 1991).

Example 2.7. (Specifiable properties)
Let φ, ψ be formulae with no temporal operators and a be a prime action symbol over the same signature. We note the ability to specify, for instance:

safety: $(\mathsf{G}\,\varphi)$ or simply φ [1];
liveness: $(\varphi \Rightarrow (\mathsf{F}\,\psi))$;
fairness:

justice: $((\mathsf{F}(\mathsf{G}\diamond a)) \Rightarrow (\mathsf{G}(\mathsf{F}\,\triangledown a)))$;
compassion: $((\mathsf{G}(\mathsf{F}\diamond a)) \Rightarrow (\mathsf{G}(\mathsf{F}\,\triangledown a)))$;

valuation: $(\triangledown a \Rightarrow (\mathsf{X}\,\varphi))$. □

[1] This simplification is possible because we are in a floating framework. Note that, by definition, a life-cycle satisfies φ if and only if it satisfies it at each $i \in I\!N$.

Example 2.8. (Activity)
Suppose we wanted to specify an active system in the sense that it should not remain idle from a certain instant on unless, in fact, it has really no chance of evolving. I.e., we want to exclude from its allowed life-cycles all those where, from a certain instant on, nothing occurs despite there being enabled actions recurrently.

Let Σ be such that $\Sigma_{act} = \{a_1, \ldots, a_k\}$.

In that case, the corresponding requirement could be specified by the axiom:

$$((\mathsf{G}(\mathsf{F}(\diamond a_1 \vee \ldots \vee \diamond a_k))) \Rightarrow (\mathsf{G}(\mathsf{F}(\triangledown a_1 \vee \ldots \vee \triangledown a_k)))). \qquad \square$$

Example 2.9. (Clock bomb)
Suppose we wanted to specify the behaviour of a clock bomb system. The idea would be to have some internal clock doing "tic" until, sometime in the future, the bomb explodes. We could chose the specification to be $clock\_bomb = (\Sigma_{clock\_bomb}, \Gamma_{clock\_bomb})$, with:

$\Sigma_{clock\_bomb,obs} = \{on\}$;
$\Sigma_{clock\_bomb,act} = \{tic, bang\}$;
Γ_{clock_bomb} consisting of the following 6 axioms:

1 $(\diamond tic \Leftrightarrow on)$
2 $(\diamond bang \Leftrightarrow on)$
3 $(\neg(\triangledown bang)) \Rightarrow ((on \Leftrightarrow \mathsf{X}\, on)))$
4 $(\triangledown bang \Rightarrow (\mathsf{X}(\neg on)))$
5 $(\neg(\triangledown bang \wedge \triangledown tic))$
6 $(on \Rightarrow (\mathsf{F}\, \triangledown bang))$

Axioms 1 and 2 mean, respectively, that *tic* and *bang* are enabled iff observation *on* is true. The third axiom ensures that the value of observation *on* can only be changed after a *bang* occurrence and axiom 4 requires that *on* is false after *bang* occurs. The fifth axiom prevents *tic* and *bang* from occurring simultaneously and the last one requires that, once turned *on*, the clock bomb will eventually *bang*.

Consider the following life-cycles over Σ_{clock_bomb}:

(i) λ such that

$\lambda_0 = \{on, \diamond tic, \diamond bang, \triangledown tic\}$,
$\lambda_1 = \{on, \diamond tic, \diamond bang, \triangledown bang\}$,
$\lambda_n = \emptyset$ for every $n > 1$;

(ii) λ' such that

$\lambda'_n = \{on, \diamond tic, \diamond bang, \triangledown tic\}$ for every $n < 50$,
$\lambda'_{50} = \{on, \diamond tic, \diamond bang, \triangledown bang\}$,
$\lambda'_n = \emptyset$ for every $n > 50$.

(iii) λ'' such that

$\lambda''_n = \{on, \diamond tic, \diamond bang, \triangledown tic\}$ for every $n \in I\!N$;

Clearly, both λ and λ' satisfy *clock_bomb* but λ'' does not satisfy axiom 6. $\square$

This is clearly a situation in which the liveness property specified by axiom 6 indicates that the specifier did not want to commit himself with the precise instant in which *bang* will occur. It is therefore making use of the inherent non-determinism on the satisfaction of eventualities as a form of underspecification. But eventualities are also used with different purposes. Consider, for instance, the following specification.

EXAMPLE 2.10. (Mutual exclusion)
Suppose we wanted to specify a protocol for maintaining mutual exclusion for two processes in the access to some critical section (CS). Each process cyclicly leaves a non-critical "idle" state becoming willing to enter the CS. Upon a certain time awaiting it must eventually enter and then leave the CS, again becoming "idle". Of course, the processes are not allowed to be at the CS at the same time. We could use the specification $mutex = \langle \Sigma_{mutex}, \Gamma_{mutex} \rangle$, with:

$\Sigma_{mutex,obs} = \{idle_1, idle_2, critical_1, critical_2\}$;
$\Sigma_{mutex,act} = \{willing_1, willing_2, in_1, in_2, out_1, out_2\}$;
Γ_{mutex} consisting of the following axioms

1 $(idle_1 \Rightarrow (\neg\, critical_1))$
2 $(idle_2 \Rightarrow (\neg\, critical_2))$
3 $(\diamond willing_1 \Leftrightarrow idle_1)$
4 $(\diamond willing_2 \Leftrightarrow idle_2)$
5 $(\diamond in_1 \Leftrightarrow ((\neg\, idle_1) \wedge (\neg\, critical_1)))$
6 $(\diamond in_2 \Leftrightarrow ((\neg\, idle_2) \wedge (\neg\, critical_2)))$
7 $(\diamond out_1 \Leftrightarrow critical_1)$
8 $(\diamond out_2 \Leftrightarrow critical_2)$
9 $(\triangledown willing_1 \Rightarrow (\mathsf{X}((\neg\, idle_1) \wedge (\neg\, critical_1))))$
10 $(\triangledown willing_2 \Rightarrow (\mathsf{X}((\neg\, idle_2) \wedge (\neg\, critical_2))))$
11 $(\triangledown in_1 \Rightarrow (\mathsf{X}\, critical_1))$
12 $(\triangledown in_2 \Rightarrow (\mathsf{X}\, critical_2))$
13 $(\triangledown out_1 \Rightarrow (\mathsf{X}\, idle_1))$
14 $(\triangledown out_2 \Rightarrow (\mathsf{X}\, idle_2))$
15 $(((\neg \triangledown willing_1) \wedge ((\neg \triangledown in_1) \wedge (\neg \triangledown out_1))) \Rightarrow$
$(((\mathsf{X}\, idle_1) \Leftrightarrow idle_1) \wedge ((\mathsf{X}\, critical_1) \Leftrightarrow critical_1)))$
16 $(((\neg \triangledown willing_2) \wedge ((\neg \triangledown in_2) \wedge (\neg \triangledown out_2))) \Rightarrow$
$(((\mathsf{X}\, idle_2) \Leftrightarrow idle_2) \wedge ((\mathsf{X}\, critical_2) \Leftrightarrow critical_2)))$
17 $(\triangledown willing_1 \Rightarrow (\mathsf{F} \triangledown out_1))$
18 $(\triangledown willing_2 \Rightarrow (\mathsf{F} \triangledown out_2))$
19 $(\neg(critical_1 \wedge critical_2))$

Axioms 1 to 18 are paired and correspond to each of the processes separately. The first two state that an idle process is not in the CS. Axioms 3 and 4 declare that a process can become willing to enter the CS iff it is idle, and axioms 5 and 6 that a process can enter the CS iff it is outside and not idle. The next two axioms say that a process can leave the CS iff it is in there. Axioms 9 to 14 specify the effect of action occurrences. Axioms 9 and 10 say that after becoming willing a process is neither idle nor in the CS, axioms 11 and 12 say that after entering a process is in the CS and axioms 13 and 14 ensure that after leaving the CS a process becomes idle. Axioms 15 and 16 express a frame rule saying that only actions can change observations. Axioms 17 and 18 express the fairness requirement for each of the processes: a process which becomes willing to enter the CS will eventually leave it. Axiom 19 is where the mutual exclusion is specified. □

3. Which operational semantics?

If we adopted a class of labelled transition systems whose infinite computations were extracted by just requiring sequentiality, we would have no way of supporting, for instance, the satisfaction of fairness properties. One may want to specify systems where, to some of its actions no fairness is required or required only on certain situations. This is often achieved by taking a model whose acceptance condition involves some additional properties on the set of states visited in each computation, eg. Büchi, Muller or Streett automata (Thomas, 90). However, for representing reactive systems it is usually more convenient to force these additional acceptance constraints on the basis of transitions rather than on states. It is the case of the fair transition systems used in (Manna and Pnueli, 1992), (Manna and Pnueli, 1993). For instance, if so specified, the treatment of fairness allows the exclusion of computations where a stuttering transition is persistently performed while other actions are enabled as shown in a previous example. Apart from all this, the operational domain we are looking for must also be such that the exact behaviour of a specified system may be synthesized from the specification and used as a prototype for execution. We also need a mandatory stuttering transition (for delays) in order to achieve, to a certain extent, the possibility of composing systems.

3.1. G-AUTOMATA

The discussion above motivates the description of our operational semantics domain, chosen in order to provide the ability to cope with the specification of fairness and liveness properties, and synthesis from a given specification. The structure of G-automata is, essentially, that of labelled transition systems with stuttering. However, we additionally incorporate a set of goals to be achieved at each of its states and a goal fulfilment mechanism to each of its transitions. By a state having a goal to be achieved we intuitively mean that the acceptable computations (sequences of transitions) from that state on must be only those which eventually include a transition fulfilling it. That is, we incorporate the mechanism for satisfying eventualities into the acceptance condition of G-automata.

DEFINITION 3.1. (G-automaton)
A *G-automaton* is a tuple $aut = (\Sigma, S, \Delta, v, G, p, f)$ consisting of:

(i) a *signature* Σ of observation and prime action symbols;
(ii) a set S of *states*;
(iii) a set $\Delta \subseteq S \times 2^{\Sigma_{act}} \times S$ of *transitions*, such that for each $e \in 2^{\Sigma_{act}}$ and $s, r \in S$, $\langle s, e, r\rangle \in \Delta$ means that there is a transition labelled by e from state s to state r;
(iv) a *state valuation map* $v : S \to 2^{\Sigma_{obs}}$, assigning to each state its set of observations;
(v) a set G of *goals*;
(vi) a *pending goals map* $p : S \to 2^G$, defining the set of pending goals at each state;
(vii) a *fulfilled goals map* $f : \Delta \to 2^G$, mapping each transition onto the set of goals it fulfils

such that the following conditions hold:

open-nature: $\langle s, \emptyset, s\rangle \in \Delta$ for each $s \in S$;
fulfilment vs. pending goals: $f(\langle s, e, r\rangle) \subseteq p(s)$ for each $\langle s, e, r\rangle \in \Delta$;
goal persistence: $(p(s) \setminus p(r)) \subseteq f(\langle s, e, r\rangle)$ for each $\langle s, e, r\rangle \in \Delta$. □

As must be clear, transitions are labelled by sets of prime action symbols instead of being labelled with only one prime action symbol. This is due to our interest in having a semantic domain which is rich enough to support, for instance, parallel composition constructs. In that case, if action symbols are meant to synchronize, our choice turns to be quite natural. Thus, from now on, we shall refer to sets of prime action symbols just as "actions".

To explain the three conditions imposed on the definition of G-automaton we have to assume a pragmatic point of view. In fact, since we consider an action to be a set of prime action symbols (meaning that all those prime actions occur from an instant in time to the following, performing a concurrent step), we have to decide what we mean by the $\emptyset$ action. We claim that it should be seen as an "invisible" change of state and that it can play the role of a stuttering transition. This choice is the key to the achievement of suitable support for hiding and interleaved parallel composition. Besides, the information introduced by the maps p and f concerning goals should be such that only pending goals are fulfilled, and that every pending goal must propagate to subsequent states, at least until some transition indeed fulfils it.

Given a G-automaton, it is possible to infer which transitions leaving a certain state may help to the eventual achievement of a pending goal. The intuition is that a transition contributes to the achievement of the goal if it is the first transition in a finite non-cyclic path of consecutive transitions leading to one which actually fulfils the goal.

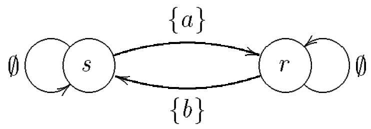
In the sequel, each transition $\delta = \langle s, e, r\rangle$ of a G-automaton shall be also referred to by $s \xrightarrow{e} r$ and we shall assume $dom(\delta) = s$, $act(\delta) = e$ and $cod(\delta) = r$.

DEFINITION 3.2. (Contribution to goal achievement)
Let $aut = (\Sigma, S, \Delta, v, G, p, f)$ be a G-automaton. We say that $\delta \in \Delta$ *contributes to the achievement of* $g \in p(s)$ iff $dom(\delta) = s$ and there exists a finite sequence of transitions $\delta_1 ... \delta_n$ such that the following conditions are satisfied:

sequentiality: $dom(\delta_{i+1}) = cod(\delta_i)$ for every $i \in \{1, ..., n-1\}$;
initialization: $\delta_1 = \delta$;
inner-cycle absence: $cod(\delta_i) \neq dom(\delta)$ for every $i \in \{1, ..., n-1\}$ and
$i \neq j$ implies $cod(\delta_i) \neq cod(\delta_j)$ for every $i, j \in \{1, ..., n\}$;
achievement: $g \in f(\delta_n)$.

We denote by $Contrib(\delta)$ the set of pending goals at $dom(\delta)$ to whose achievement δ contributes. □

Example 3.3. (Contribution)
Consider the G-automaton aut' over signature Σ', such that $\Sigma'_{obs} = \{x\}$ and $\Sigma'_{act} = \{a, b\}$, represented by the diagram



and with:

$v'(s) = \emptyset$, $v'(r) = \{x\}$;
$G' = \{g'\}$;
$p'(s) = p'(r) = \{g'\}$;
$f'(r \xrightarrow{\{b\}} s) = \{g'\}$, $f'(s \xrightarrow{\emptyset} s) = f'(r \xrightarrow{\emptyset} r) = f'(s \xrightarrow{\{a\}} r) = \emptyset$.

For each of the transitions of aut we have:

$Contrib(s \xrightarrow{\emptyset} s) = \emptyset$ (every sequence ending up in $r \xrightarrow{\{b\}} s$ has an inner-cycle);
$Contrib(r \xrightarrow{\emptyset} r) = \emptyset$ (every sequence ending up in $r \xrightarrow{\{b\}} s$ has an inner-cycle);
$Contrib(s \xrightarrow{\{a\}} r) = \{g'\}$ (consider the sequence $s \xrightarrow{\{a\}} r \xrightarrow{\{b\}} s$);
$Contrib(r \xrightarrow{\{b\}} s) = \{g'\}$ (consider the sequence $r \xrightarrow{\{b\}} s$). □

Having defined G-automata, we now face the problem of defining suitable relationships between them. We are looking for an adequate notion of homomorphism between G-automata, which may be seen as a guarantee for partial simulation and preservation of the goal achievement structure. The definition was motivated by those in (Caleiro, 1993), (Manna and Pnueli, 1992), (Sernadas, Costa and Sernadas, 1994).

Definition 3.4. (G-automata homomorphism)
Let $aut = (\Sigma, S, \Delta, v, G, p, f)$ and $aut' = (\Sigma', S', \Delta', v', G', p', f')$ be two G-automata. A *homomorphism* $h : aut \to aut'$ is a 4-tuple $h = (h_{obs}, h_{act}, h_S, h_G)$, where $h_{obs} : \Sigma'_{obs} \to \Sigma_{obs}$, $h_{act} : \Sigma'_{act} \to \Sigma_{act}$, $h_S : S \to S'$ and $h_G : G' \to G$ are set maps such that the following structure preservation conditions are satisfied:

partial simulation:
$\langle h_S(s), h_{act}^{-1}(e), h_S(r)\rangle \in \Delta'$ for each $\langle s, e, r\rangle \in \Delta$;
observational correspondence:
$h_{obs}^{-1}(v(s)) \subseteq v'(h_S(s))$ for each $s \in S$;
pending goals correspondence:
$h_G(p'(h_S(s))) \subseteq p(s)$ for each $s \in S$;
goal fulfilment correspondence:
$h_G(f'(\langle h_S(s), h_{act}^{-1}(e), h_S(r)\rangle)) \subseteq f(\langle s, e, r\rangle)$ for each $\langle s, e, r\rangle \in \Delta$. □

Intuitively, the *partial simulation* condition of an homomorphism can be seen as a way of representing a behavioural relationship between a system and one of its components.

In fact, if we regard the source of the homomorphism as a whole system and the target as a component, the condition requires that every transition of the whole system has a counterpart in terms of a corresponding transition of the component. The same intuition applies to the *observational correspondence* condition, since it states that any observation in a state of the whole system must be supported by the relevant observations in the corresponding state of the component. The *pending goals* and *goal fulfilment correspondence* conditions force pending goals and goal fulfilment performance of a component to extend to corresponding states and transitions of the whole.

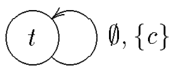
To illustrate some of the potentialities of this semantic domain, we shall take a look at its ability to support simple parallel composition with preservation of the goal mechanisms of the parts involved.

DEFINITION 3.5. (Parallel composition of G-automata)
The parallel composition $aut' \parallel aut''$ of G-automata $aut' = (\Sigma', S', \Delta', v', G', p', f')$ and $aut'' = (\Sigma'', S'', \Delta'', v'', G'', p'', f'')$ is $(\Sigma, S, \Delta, v, G, p, f)$, where:

(i) $\Sigma = (\Sigma'_{obs} + \Sigma''_{obs}, \Sigma'_{act} + \Sigma''_{act})$;
(ii) $S = S' \times S''$;
(iii) $\Delta = \{\langle\langle s', s''\rangle, e' + e'', \langle r', r''\rangle\rangle : \langle s', e', r'\rangle \in \Delta', \langle s'', e'', r''\rangle \in \Delta''\}$;
(iv) $v(\langle s', s''\rangle = v'(s') + v''(s'')$ for every $s' \in S'$ and $s'' \in S''$;
(v) $p(\langle s', s''\rangle = p'(s') + p''(s'')$ for every $s' \in S'$ and $s'' \in S''$;
(vi) $\phi(\langle\langle s', s''\rangle, e' + e'', \langle r', r''\rangle\rangle) = f'(\langle s', e', r'\rangle) + f''(\langle s'', e'', r''\rangle)$ for every $(s', e', r') \in \Delta'$ and $(s'', e'', r'') \in \Delta''$. □

Note that $+$ stands for the disjoint union and $\times$ for the cartesian product of sets. Moreover, parallel composition is a product in the corresponding category[1] of G-automata (Caleiro, 1995).

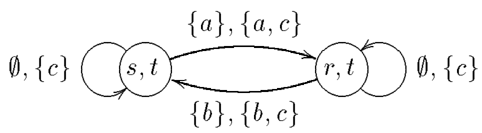
EXAMPLE 3.6. (Parallel composition)
Let aut' be the automaton of example 3.3 and aut'' be the G-automaton over signature Σ'', such that $\Sigma''_{obs} = \emptyset$ and $\Sigma''_{act} = \{c\}$, represented by



and with:

$$v''(t) = \emptyset;\ G'' = \{g''\};\ p''(t) = \{g''\};\ f''(t \xrightarrow{\{c\}} t) = \{g''\},\ f''(t \xrightarrow{\emptyset} t) = \emptyset.$$

Then, $aut' \parallel aut''$ is the G-automaton over $\langle\{x\}, \{a, b, c\}\rangle$ with shape



[1] Refer to (Adámek, Herrlich and Strecker, 1990) for a textbook on category theory.

and with:

$$
\begin{aligned}
&v(\langle s,t\rangle)=\emptyset,\ v(\langle r,t\rangle)=\{x\};\ G=\{g',g''\};\ p(\langle s,t\rangle)=p(\langle r,t\rangle)=\{g',g''\};\\
&f(\langle s,t\rangle \xrightarrow{\emptyset} \langle s,t\rangle)=f(\langle r,t\rangle \xrightarrow{\emptyset} \langle r,t\rangle)=f(\langle s,t\rangle \xrightarrow{\{a\}} \langle r,t\rangle)=\emptyset,\\
&f(\langle r,t\rangle \xrightarrow{\{b\}} \langle s,t\rangle)=\{g'\},\\
&f(\langle s,t\rangle \xrightarrow{\{c\}} \langle s,t\rangle)=f(\langle r,t\rangle \xrightarrow{\{c\}} \langle r,t\rangle)=f(\langle s,t\rangle \xrightarrow{\{a,c\}} \langle r,t\rangle)=\{g''\},\\
&f(\langle r,t\rangle \xrightarrow{\{b,c\}} \langle s,t\rangle)=\{g',g''\}.
\end{aligned}
$$

□

Introducing goals into transition systems was brought up as a way of restricting the usual notion of computation, imposing necessary fulfilment of pending goals at ongoing states. Due to this fact we shall restrict our attention to goal-consistent G-automata i.e., G-automata where every pending goal can be fulfilled.

If Δ is the set of transitions of a G-automaton and s is one of its states, we denote by $\Delta(s)$ the set $\{\delta \in \Delta : dom(\delta) = s\}$.

Definition 3.7. (Goal-consistent G-automaton)
We say a G-automaton $aut = (\Sigma, S, \Delta, v, G, p, f)$ is *goal-consistent* iff:

for every $s \in S$ and $g \in p(s)$ there exists $\delta \in \Delta(s)$ s.t. $g \in Contrib(\delta)$. □

Example 3.8. (Goal consistency)
The G-automaton of example 3.3 is clearly goal-consistent. It would not be, however, if we removed from it the transition $\delta = \langle s, \{a\}, r\rangle$. Note that $g' \in p'(s)$ and δ is the only transition with domain s s.t. $g' \in Contrib(\delta)$. □

The following result shows that goal-consistency is preserved by parallel composition.

Proposition 3.9. (Goal-Consistency of Parallel Composition)
The parallel composition $aut' \parallel aut''$ of G-automata is goal-consistent if and only if both aut' and aut'' are goal-consistent. □

3.2. Computations and satisfaction

A computation of a G-automaton is, as usual, a sequence of consecutive transitions. However, it is possible that some of these sequences do not correspond to our intended idea of goal fulfilment. In that case, we have to filter the undesired ones by imposing that pending goals are fulfilled. Let us consider fixed a G-automaton $aut = (\Sigma, S, \Delta, v, G, p, f)$.

Definition 3.10. (Computation)
A *computation* of *aut* is a map $\eta : \mathbb{N} \to \Delta$ such that:

(i) $dom(\eta_{i+1}) = cod(\eta_i)$ for every $i \in \mathbb{N}$;
(ii) for each $i \in \mathbb{N}$, if $g \in p(dom(\eta_i))$ then exists $j \geq i$ such that $g \in f(\eta_j)$.

We denote by $Comp_{aut}$ the set of all computations of *aut*. □

The definition of formula satisfaction by a G-automaton is similar to the above defined satisfaction by life-cycle. Obviously, a life-cycle can be extracted from each computation of an automaton. Consequently, a life-structure can be extracted by considering the set of all life-cycles corresponding to its computations.

DEFINITION 3.11. (Menu of a state)
The *menu of state* $s \in S$ is the set $Menu(s) = \bigcup_{\delta \in \Delta(s)} act(\delta)$. □

The menu of a state is the set of actions whose occurrence is enabled in that state.

DEFINITION 3.12. (Life-cycle induced by computation)
Let $\eta \in Comp_{aut}$. The *life-cycle induced by* η is the sequence $Lfc(\eta)$ such that $Lfc(\eta)_i = v(dom(\eta_i)) \cup \{\diamond a : a \in Menu(dom(\eta_i))\} \cup \{\triangledown a : a \in act(\eta_i)\}$ for each $i \in I\!N$. □

EXAMPLE 3.13. (Computations and life-cycles)
Once again considering the G-automaton aut' of example 3.3, clearly:

(i) η such that, for each $n \in I\!N$,

$\eta_n = \langle s, \emptyset, s\rangle$, is not a computation of aut';

(ii) η' such that, for each $n \in I\!N$,

$\eta'_{2n} = \langle s, \{a\}, r\rangle$ and $\eta'_{2n+1} = \langle r, \{b\}, s\rangle$ is a computation of aut'
with
$Lfc(\eta')_{2n} = \{\diamond a, \triangledown a\}$ and $Lfc(\eta')_{2n+1} = \{x, \diamond b, \triangledown b\}$. □

PROPOSITION 3.14. (G-automata and life-structures)
The set $Lfs(aut) = \{Lfc(\eta) : \eta \in Comp_{aut}\}$ is a life-structure. □

DEFINITION 3.15. (Satisfaction by G-automaton)
We say that aut satisfies $\varphi \in Form_\Sigma$, $aut \models \varphi$, iff $Lfs(aut) \models \varphi$. Analogously, aut satisfies the set $\Gamma \subseteq Form_\Sigma$, $aut \models \Gamma$, iff $aut \models \varphi$ for each $\varphi \in \Gamma$, and aut satisfies a specification $spec = (\Sigma, \Gamma)$, $aut \models spec$, iff $aut \models \Gamma$. □

4. Synthesizing G-automata from specifications

In this section we aim at presenting an algorithmic construction method for the operational semantics (suitable goal-consistent G-automaton) of a given specification over a finite signature (finite sets of observation and prime action symbols). By suitable we mean not only that it must satisfy the specification but also that it must fully represent any other automaton satisfying the specification. I.e., the synthesized G-automaton must be in some sense canonical among all those G-automata which satisfy the specification. The synthesis algorithm presented capitalizes on the results in (Caleiro, 1993) and consists of an adapted version of the tableaux-based decision procedure for propositional temporal logic presented in (Ben-Ari, Pnueli and Manna, 1983) and, specifically, of the synthesis method proposed in (Manna and Wolper, 1984). We start by introducing the envisaged tableaux system (Bell and Machover, 1977) for the temporal logic we are using. Insight was also given by similar synthesis approaches for computation tree logics and producing tree automata (Clarke and Emerson, 1981), (Pnueli and Rosner, 1988).

4.1. Tableaux for temporal logic

A tableau is a finite tree whose nodes are labelled by sets of formulae. Given a set of tree formation (inference) rules we can build a tableau whose root node is any given finite set of formulae. The rules we are interested in are similar to those used both in (Ben-Ari, Pnueli and Manna, 1983) and (Manna and Wolper, 1984) and are based on the fixed point properties of temporal operators. The idea behind them is to make the suitable requirements over the next instant in order to satisfy a formula at the present.

Definition 4.1. (Inference rules schemata, inference rules, premises, conclusion) Let φ,ψ be formulae. The *inference rules schemata* are:

$$\begin{array}{c}(\neg(\neg\varphi))\\ {\scriptstyle(\neg\neg)}\ \Big|\\ \varphi\end{array}\qquad\qquad \begin{array}{ccc} & (\varphi\Rightarrow\psi) & \\ {\scriptstyle(\Rightarrow)}\ / & & \backslash \\ (\neg\varphi) & & \psi\end{array}\qquad\qquad \begin{array}{c}(\neg(\varphi\Rightarrow\psi))\\ {\scriptstyle(\neg\Rightarrow)}\ \Big|\\ \varphi\\ (\neg\psi)\end{array}$$

$$\begin{array}{c}(\neg(\mathsf{X}\,\varphi))\\ {\scriptstyle(\neg\mathsf{X})}\ \Big|\\ (\mathsf{X}(\neg\varphi))\end{array}\qquad\qquad \begin{array}{ccc} & (\varphi\,\mathsf{U}\,\psi) & \\ {\scriptstyle(\mathsf{U})}\ / & & \backslash \\ \psi & & \begin{array}{c}\varphi\\ (\mathsf{X}(\varphi\,\mathsf{U}\,\psi))\end{array}\end{array}\qquad\qquad \begin{array}{ccc} & (\neg(\varphi\,\mathsf{U}\,\psi)) & \\ {\scriptstyle(\neg\mathsf{U})}\ / & & \backslash \\ \begin{array}{c}(\neg\psi)\\ (\neg\varphi)\end{array} & & \begin{array}{c}(\neg\psi)\\ (\mathsf{X}(\neg(\varphi\,\mathsf{U}\,\psi)))\end{array}\end{array}$$

An *inference rule* is any concrete instance of an inference rule schema. Given an unary inference rule

$$\begin{array}{c}\theta\\ \Big|\\ \alpha_1\\ \ldots\\ \alpha_n\end{array}$$

we say that $\alpha_1, \ldots, \alpha_n$ is the premise of the rule and that θ is its conclusion. Similarly, given a binary inference rule

$$\begin{array}{ccc} & \theta & \\ / & & \backslash \\ \begin{array}{c}\alpha_1\\ \ldots\\ \alpha_n\end{array} & & \begin{array}{c}\beta_1\\ \ldots\\ \beta_m\end{array}\end{array}$$

$\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_m$ are the premises of the rule and θ is its conclusion. □

The above defined inference rules schemata for the primary propositional logic connectives and temporal operators can be used to derive rules for those other logical connectives and temporal operators introduced by abbreviation in subsection 2.1. We get:

$$(\vee)\ \frac{(\varphi \vee \psi)}{\varphi \qquad \mid \qquad \psi} \qquad\qquad (\wedge)\ \frac{(\varphi \wedge \psi)}{\begin{array}{c}\varphi\\ \psi\end{array}} \qquad\qquad (\Leftrightarrow)\ \frac{(\varphi \Leftrightarrow \psi)}{\begin{array}{c}\varphi\\ \psi\end{array} \quad \mid \quad \begin{array}{c}(\neg\,\varphi)\\ (\neg\,\psi)\end{array}}$$

$$(\neg\,\vee)\ \frac{(\neg(\varphi \vee \psi))}{\begin{array}{c}(\neg\,\varphi)\\ (\neg\,\psi)\end{array}} \qquad\qquad (\neg\,\wedge)\ \frac{(\neg(\varphi \wedge \psi))}{\neg\,\varphi \qquad \mid \qquad \neg\,\psi} \qquad\qquad (\neg\,\Leftrightarrow)\ \frac{(\neg(\varphi \Leftrightarrow \psi))}{\begin{array}{c}\varphi\\ (\neg\,\psi)\end{array} \quad \mid \quad \begin{array}{c}(\neg\,\varphi)\\ \psi\end{array}}$$

$$(\mathsf{G})\ \frac{(\mathsf{G}\,\varphi)}{\begin{array}{c}\varphi\\ (\mathsf{X}(\mathsf{G}\,\varphi))\end{array}} \qquad\qquad (\neg\,\mathsf{G})\ \frac{(\neg(\mathsf{G}\,\varphi))}{(\neg\,\varphi) \qquad \mid \qquad (\mathsf{X}(\neg(\mathsf{G}\,\varphi)))}$$

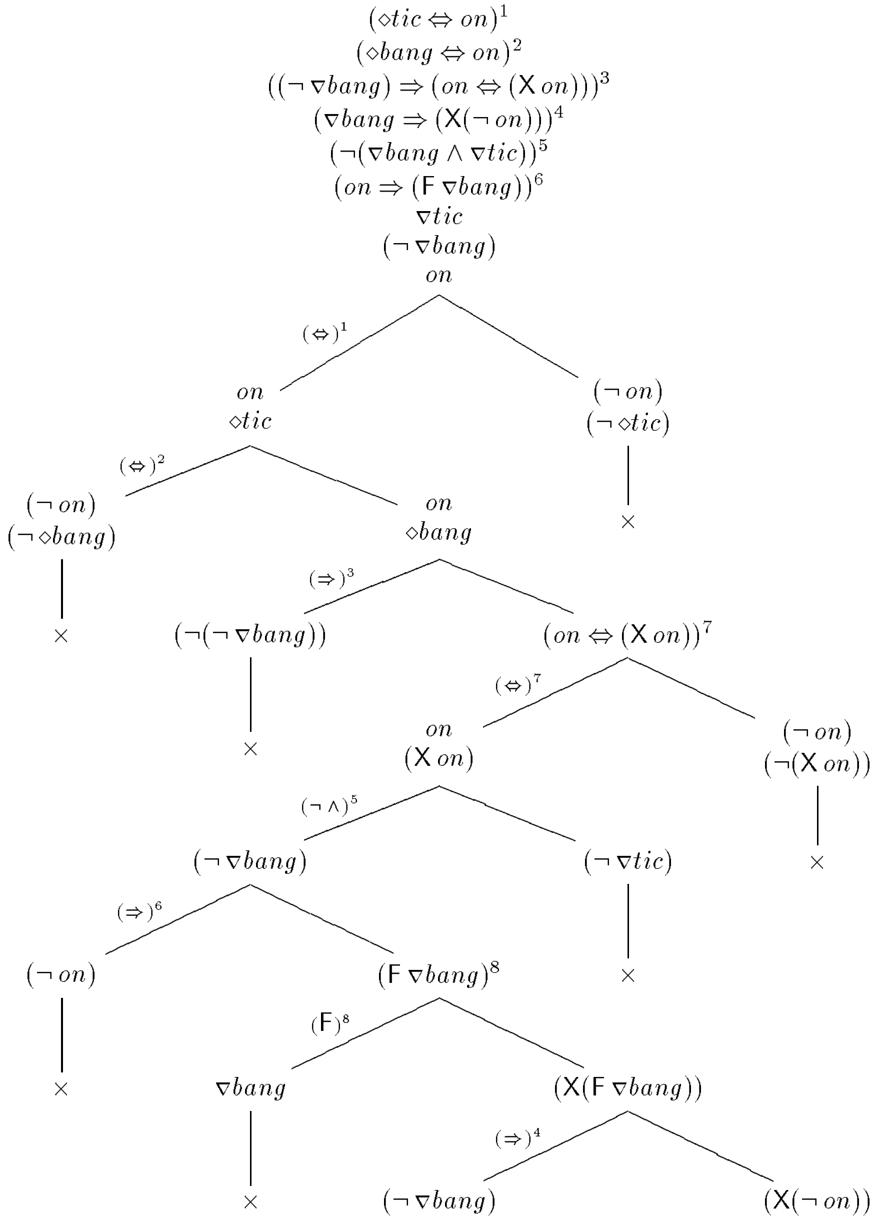
$$(\mathsf{F})\ \frac{(\mathsf{F}\,\varphi)}{\varphi \qquad \mid \qquad (\mathsf{X}(\mathsf{F}\,\varphi))} \qquad\qquad (\neg\,\mathsf{F})\ \frac{(\neg(\mathsf{F}\,\varphi))}{\begin{array}{c}(\neg\,\varphi)\\ (\mathsf{X}(\neg(\mathsf{F}\,\varphi)))\end{array}}$$

We can now take a look at how inference rules are used to build tableaux. Let us first introduce some useful notions. A branch of a finite tree to be any finite sequence of nodes $\Psi_0, \ldots, \Psi_n$ such that Ψ_0 is the root of the tree, each Ψ_{i+1} is a successor of Ψ_i, and Ψ_n has no successors. We say Ψ_n is the final node of the branch. We also define a leaf of a finite tree to be the final node of some of its branches. We say that a branch is contradictory if its nodes contain some formula φ and its negation $(\neg\,\varphi)$.

Definition 4.2. (Tableaux)
Let Φ be a finite set of formulae over Σ. The tree with only one node consisting of Φ is a *tableau* for Φ. If T is a *tableau* and the conclusion of an inference rule is in a branch of T then T', the tree which results from adding nodes corresponding to each premise of the rule as successors of the final node of the branch of T considered, is a *tableau*. We write $T \prec T'$. An *exhausted tableau* for Φ is a tableau for Φ such that each of its branches is either contradictory or no further rules can be applied. □

As a simplification, we assume that no rule is applied twice to the same formula in the same branch and that no rules are applied to contradictory branches. Note also that, by definition, every tableau is a finite tree.

EXAMPLE 4.3. (Tableau)
Recall the *clock_bomb* specification in example 2.9. A tableau for the set of formulae $\Gamma_{clock\_bomb} \cup \{on, \triangledown tic, (\neg\, \triangledown bang)\}$ is:



From each parent node to its sons we mention the rule schemata which is being used and we number the formula to which it is being applied. The crosses $\times$ denote contradictory branches. We conclude that the specification axioms plus the formulae *on*, $\triangledown tic$ and $(\neg\, \triangledown bang)$ can only hold if in the next instant *on* and $(\mathsf{F}\, \triangledown bang)$ hold. $\square$

PROPOSITION 4.4. (Soundness)
A life-cycle λ over Σ satisfies at $i \in I\!N$ the conclusion of an inference rule iff it satisfies all the formulae in some of its premises. A life-cycle λ over signature Σ satisfies at $i \in I\!N$ all the formulae in the root of a tableau iff it satisfies all the formulae in some of its branches. □

As immediate corollaries we have that the root of a tableau is inconsistent (not satisfiable) if and only if all its branches are inconsistent. In particular, if all its branches are contradictory then its root is inconsistent.

The following result shows the existence of a procedure *tableau* which constructs an exhausted tableau given the set of formulae wanted in its root.

PROPOSITION 4.5. (Existence of tableaux construction procedure)
Let Φ be a finite set of formulae over Σ. There is no infinite sequence $\{T_i\}_{i \in I\!N}$ of tableaux for Φ s.t. $T_i \prec T_{i+1}$ for every $i \in I\!N$. Moreover, the set of tableaux for Φ is finite and each of them can be effectively computed. □

The proof of the result above is straightforward, although dull, and can be done by induction on the structure of the formulae in Φ.

We also consider available a procedure *branches* which takes a tableau as input and produces the set of sets of formulae corresponding to each of its non-contradictory branches. From now on we shall call branch to the set of all the formulae which occur in its nodes.

4.2. THE SYNTHESIS ALGORITHM

The set of inference rules we use is such that a tableau is a way of requiring (when necessary) that the "next state" conditions hold. We simply get to knowing which formulae must be satisfied in the next instant in order to satisfy the actual set at the present. This is the overall idea behind the synthesis method. It proceeds by checking which sets of formulae may characterize a state reachable from some other by the occurrence of a certain set of prime actions. The procedure shall, however, require that the working signature is finite i.e., that both the sets of observation and prime action symbols are finite. Let Σ be such a signature.

We introduce some auxiliary notation:

(i) if $B \subseteq \Sigma_{obs}$, $obs(B) = B \cup \{(\neg\, b) : b \in (\Sigma_{obs} \setminus B)\}$;
(ii) if $e \in 2^{\Sigma_{act}}$, $\diamond(e) = \{\diamond a : a \in e\} \cup \{(\neg\, \diamond a) : a \in (\Sigma_{act} \setminus e)\}$;
(iii) if $e \in 2^{\Sigma_{act}}$, $\nabla(e) = \{\nabla a : a \in e\} \cup \{(\neg\, \nabla a) : a \in (\Sigma_{act} \setminus e)\}$;
(iv) if $\Phi \subseteq Form_\Sigma$, $next(\Phi) = \{\varphi : (\mathsf{X}\, \varphi) \in \Phi\}$;
(v) if $\Phi \subseteq Form_\Sigma$, $\mathsf{G}(\Phi) = \{(\mathsf{G}\, \varphi) : \varphi \in \Phi\}$.

Let us now consider fixed a specification $spec = (\Sigma, \Gamma)$ over the finite signature Σ.

For the sake of simplicity, we shall prefix the specification formulae with a G operator. This simplifies the method because we are using a floating version of temporal logic but a state by state reasoning. Anyway, it is easy to see that a life-cycle λ satisfies a formula φ iff λ satisfies $(\mathsf{G}\, \varphi)$ (remember that the logic is floating).

The synthesis method consists of five steps. First, given the input specification, and by tableau constructions, we build a directed labelled graph with sets of formulae as nodes and actions as edge labels. We also obtain a candidate valuation map by looking at observations in each of the nodes. Step two is a filter for non-open nodes. Our aim is to get a G-automaton, thus all the states must satisfy the open-nature condition. The graph obtained in step one is restricted to those nodes which have an empty-set edge to themselves. The valuation map is also restricted. The third step, also a filtering step, deals with our notion of enabling. A node establishes its enablings if for each prime action whose enabling atom is in the node, there exists an edge leaving it whose label contains that prime action. We get a yet more restricted graph containing only those nodes which establish their enablings, and corresponding edges. Once again we confine the valuation map to the nodes left. Steps four and five have to do with goals. In step four we build the set of goals and the pending goals and goal fulfilment maps. This is done by analysis of "eventuality" formulae throughout the obtained graph. In step five we check for goal-consistency of the states of the G-automaton obtained after step four.

4.2.1. Step 1 - Graph construction

To produce the underlying candidate graph of the claimed G-automaton we proceed by considering that, at each state, the set of axioms of the specification plus some set of enablings and observations must hold. Then, by applying a tableau construction to each such state on considering the execution of each of the allowed sets of prime actions and by looking at the X formulae appearing in each of its non-contradictory branches we build up new corresponding states. The set S^1 of state candidates is inductively defined as follows:

(i) $\mathsf{G}(\Gamma) \cup \diamond(e) \cup obs(B) \in S^1$ for each $e \in 2^{\Sigma_{act}}$ and $B \subseteq \Sigma_{obs}$;
(ii) $next(\Omega) \cup \diamond(e) \cup obs(B) \in S^1$ for each $e \in 2^{\Sigma_{act}}$ and $B \subseteq \Sigma_{obs}$, if exist $\Theta \in S^1$ and $d \subseteq \{a : \diamond a \in \Theta\}$ such that $\Omega \in branches(tableau(\Theta \cup \nabla(d)))$.

This set S^1 is finite and can, hence, be effectively computed. This can be proved by structural induction on the finite set of formulae of the specification by recalling that tableaux can be computed, and by using the fact that the possible X formulae in branches of tableaux and the signature are both finite. More specifically, we can give an upper bound on the number of states of S^1. Just note that its elements can only contain subformulae of the initial set of prefixed axioms $\mathsf{G}(\Gamma)$, observation and enabling atoms, and negations of those. Hence, since the set of axioms of the specification is finite, the number of subformulae of $\mathsf{G}(\Gamma)$ is also finite, say s. Moreover, the set of observation and enabling atoms has size $o + e = |\Sigma_{obs}| + |\Sigma_{act}|$. Therefore, there could be no more than $2(s + o + e)$ distinct formulae in each state and thus no more than $4^{(s+o+e)}$ states.

Each tableau construction then possibly gives rise to new candidate states corresponding to its non-contradictory branches. However, different branches may lead to the same state. We can associate with each "transition" $\delta = \langle \Theta, d, next(\Omega) \cup \diamond(e) \cup obs(B) \rangle$ the union of all the formulae in all those branches. We call to this set the extension of δ. Let $\{\Gamma_1, \ldots, \Gamma_n\} = \{\Gamma \in branches(tableau(\Theta \cup \nabla(d))) : next(\Gamma) = next(\Omega)\}$. Then:

$$Ext(\delta) = \Gamma_1 \cup \ldots \cup \Gamma_n.$$

The candidate set of transitions is:

$$\Delta^1 = \{\langle\Theta, d, next(\Omega) \cup \diamond(e) \cup obs(B)\rangle : e \in 2^{\Sigma_{act}},\ B \subseteq \Sigma_{obs},\ d \subseteq \{a : \diamond a \in \Theta\},\ \Omega \in branches(tableau(\Theta \cup \triangledown(d)))\};$$

Obviously S^1 and Δ^1 give rise to a directed graph $Graph^1 = (S^1, \Delta^1, dom, cod)$.
We can already define the candidate state valuation map v^1 by:

$$v^1(\Theta) = \Sigma_{act} \cap \Theta \text{ for each } \Theta \in S^1.$$

As was pointed out in (Manna and Wolper, 1984) the construction of this graph consists of a search for all potential life-cycles satisfying the specification. That is, every life-cycle satisfying the formulae corresponds to some infinite path in the graph and every finite path in the graph generates a prefix of some life-cycle satisfying the specification.

4.2.2. Step 2 - Open-nature checking

Since we are not interested in obtaining a graph including every possible life-cycle satisfying the axioms but just those life-cycles corresponding to computations of potential G-automata we have to get rid of those which do not fulfill the open-nature condition. Thus, we exclude from the graph all the nodes computed in 4.2.1 which do not have an emptyset labelled transition to themselves.
The set S^2 of state candidates for which the open-nature condition holds is:

$$S^2 = \{\Theta \in S^1 : \langle\Theta, \emptyset, \Theta\rangle \in \Delta^1\}.$$

The Δ^2 set of transitions is just the according subset of Δ^1:

$$\Delta^2 = \{\delta \in \Delta^1 : dom(\delta), cod(\delta) \in S^2\}.$$

The restricted graph obtained is $Graph^2 = (S^2, \Delta^2, dom, cod)$.
Analogously, the v^2 state valuation map is the restriction of v^1 to S^2:

$$v^2(\Theta) = v^1(\Theta) \text{ for each } \Theta \in S^2.$$

4.2.3. Step 3 - Enabling checking

We now ensure that enablings in each state agree with the labels of its outgoing edges.
The set T of open state candidates which do not establish their enablings is the least subset T of S^2 satisfying the condition:

if there exist $a \in \Sigma_{act}$ and $\Theta \in S^2$ s.t. $\diamond a \in \Theta$ and $\{cod(\delta) : \delta \in \Delta^2,\ dom(\delta) = \Theta,\ a \in act(\delta)\} \subseteq T$ then $\Theta \in T$.

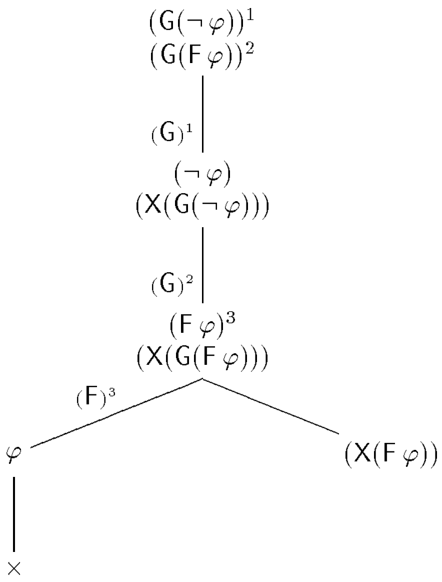
Note that since $Graph^2$ is finite this set can be effectively calculated.
We now define another approximating graph, $Graph^3 = (S^3, \Delta^3, dom, cod)$, such that:

(i) $S^3 = S^2 \backslash T$;
(ii) $\Delta^3 = \{\delta \in \Delta^2 : dom(\delta),\ cod(\delta) \in S^3\}$;
(iii) $v^3(\Theta) = v^2(\Theta)$, for each $\Theta \in S^3$.

4.2.4. Step 4 - Goal construction

By now one may wonder if the life-cycles corresponding to "computations" of $Graph^3$ necessarily satisfy the specification. To answer this question we have to take a closer look at the nature of the formulae inside each state. A priori, the answer should be affirmative. That is exactly the intention of the tableaux method. However, a problem arises with a particular kind of formulae, usually referred to as eventualities. An eventuality formula is, exactly as its name suggests, a formula whose satisfaction at a certain $i \in I\!N$ can be postponed to $i+1$. That is, a formula which eventually becomes true, but whose satisfaction may be attained as late as one wishes. By looking at the rules of our inference system (or alternatively to the definition of formula satisfaction), it is easy to see that the basic constructors of eventualities are those formulae of the kind $(\neg(\varphi \mathsf{U} \psi))$ which require the eventual satisfaction of $(\neg\varphi)$. So, in general, the answer to the question above is negative. In a state containing eventualities, it is possible that some computation starting there always performs transitions which delay their satisfaction. We therefore must get rid of such computations. This is the reason why we enriched the automata with a goal structure. Eventualities occurring in any of the candidate states give rise to pending goals which persist until some transition indeed fulfils them. If a state contains a formula of the referred kind, $(\neg\varphi)$ shall originate a goal, its fulfilment by transitions being inferred from the structure of the corresponding tableaux. Due to disjunctions, we will have to consider each goal to be a set of formulae instead of a plain formula. Note also that, by abbreviation, the pattern $(\neg(\varphi \mathsf{U} \psi))$ includes the usual "sometime in the future" formulae, $(\mathsf{F}\,\gamma) \equiv (\neg((\neg\gamma) \mathsf{U} (\gamma \wedge (\neg\gamma))))$.

Example 4.6. (Satisfaction vs. eventualities)
A specification including $(\neg\varphi)$ and $(\mathsf{F}\,\varphi)$ is clearly unsatisfiable. However, a tableau for it has non-contradictory leaves, the problem being ever postponed.



□

To start with, we shall need a way of taking a closer look at the structure of states. This can be done by a tableau, which reduces each state to its more atomic constituents. Since by definition of snapshot no prime action can occur without being enabled we just previously add the non-occurrence of every prime action which is not enabled at that state.

DEFINITION 4.7. (Properties/consequences)
The set of *properties* of $\Theta \in S^3$ is $Prop(\Theta) = \Theta \cup \{(\neg \triangledown a) : (\neg \diamond a) \in \Theta\}$ and its set of *consequences* is $Conseq(\Theta) = branches(tableau(Prop(\Theta)))$. □

We now check for eventuality patterns in the consequences of each state. The unfulfilled ones shall be called "target" formulae. These will be the building blocks of the goals at each state. To motivate this, note that several alternative "target" formulae may occur in distinct branches and thus, recalling the soundness result for tableaux, it is enough to require the full satisfaction of the "targets" in just one of them. This leads us to requiring that their disjunction is true.

DEFINITION 4.8. (Targets of a branch)
Let $\Theta \in S^3$ and $\Gamma \in Conseq(\Theta)$. We say that a formula $(\neg \varphi)$ is a *target* of branch Γ if $(\neg \varphi) \notin \Gamma$ but there exists a formula in Γ with the pattern $(\neg(\varphi \mathsf{U} \psi))$. We denote by $targets(\Gamma)$ the set of all target formulae of Γ. □

In quite the same way, using the derived temporal operators, we would have a target formulae φ for patterns $(\mathsf{F}\,\varphi)$ and a target formula $(\neg \varphi)$ for patterns $(\neg(\mathsf{G}\,\varphi))$.

We now define the set $Goals(\Theta)$ of pending goals at each $\Theta \in S^3$. For the purpose let:

(i) $Targets(\Theta) = \bigcup_{\Gamma \in Conseq(\Theta)} targets(\Gamma)$;
(ii) $Relevant(\Theta) = \{\Gamma \in Conseq(\Theta) : targets(\Gamma) \neq \emptyset\}$;
(iii) $Max(\Theta) = \{\Gamma \in Relevant(\Theta) : \{\Gamma' \in Conseq(\Theta) : targets(\Gamma) \subset targets(\Gamma')\} = \emptyset\}$.

Note that we use $\subset$ to denote strict inclusion.

DEFINITION 4.9. (Pending goals at Θ)
Let $\Theta \in S^3$ and $g \subseteq Targets(\Theta)$. We say that g is a *pending goal at* Θ, $g \in Goals(\Theta)$, iff

(i) $g \neq \emptyset$;
(ii) $g \cap targets(\Gamma) \neq \emptyset$ for every $\Gamma \in Max(\Theta)$;
(iii) if $\emptyset \neq g' \subset g$ then, for some $\Gamma \in Max(\Theta)$, $g' \cap targets(\Gamma) = \emptyset$. □

Intuitively, g intersects the targets of all the branches in the consequences of the state, but none of its subsets does. It is a minimum span of the set of targets with respect to the alternative maximal branches of $Conseq(\Theta)$.

Easily, one of the sets of relevant targets is accomplished iff all the goals are satisfied.

We can now define the set G^3 of candidate goals and the pending goals map p^3 of the candidate G-automaton:

(i) $G^3 = \bigcup_{\Theta \in S^3} Goals(\Theta)$;
(ii) $p^3(\Theta) = Goals(\Theta)$ for each $\Theta \in S^3$.

As far as goal fulfilment is concerned, the obvious idea is that to satisfy a goal it suffices the satisfaction of one of its formulae (remember that goals as sets were introduced to deal with disjunctions). The satisfaction of formulae by a transition will be established by looking at the corresponding tableau and at the properties of the corresponding reached state.

Hence, for each transition $\delta \in \Delta^3$, the goal fulfilment map f is defined by $g \in f(\delta)$ iff $g \in p(dom(\delta))$ and at least one of the following two conditions holds:

(i) $g \cap Ext(\delta) \neq \emptyset$;
(ii) $branches(tableau(Prop(cod(\delta)) \cup \{(\neg\, \varphi) : \varphi \in g\})) = \emptyset$.

The first condition is equivalent to saying that some formula in the goal is satisfied by performing transition δ. The second condition holds if some formula in the goal is a consequence of the formulae in $cod(\delta)$ i.e., the reached state. The same is to say that a transition fulfils a goal whenever some of its formulae is a consequence of the set of occurring prime actions and of the properties of its domain state or alternatively if some formula in the goal is a consequence of the properties of its codomain state.

Note that if we enrich the labelled graph structure $Graph^3$ with the set of goals G, the pending goals map p and the goal fulfilment map f (obtained as described previously) we obtain a G-automaton.

4.2.5. Step 5 - Goal-consistency checking (minimization)

The problem is now reduced to checking whether or not the obtained G-automaton is goal-consistent. This can be solved by calculating the reachability graph of each state with a pending goal and by checking if any of the transitions that fulfil that goal is reachable. We use, of course, the previously defined notion of contribution.

Definition 4.10. (G-automaton synthesized from specification)
The G-automaton *synthesized* from *spec* is $syn = (\Sigma, S, \Delta, v, G, p, f)$ such that:

(i) $S = \{\Theta \in S^3 : p^3(\Theta) \subseteq \bigcup_{\delta \in \Delta(\Theta)} Contrib(\delta)\}$;
(ii) $\Delta = \{\delta \in \Delta^3 : dom(\delta), cod(\delta) \in S\}$;
(iii) $G = \bigcup_{\Theta \in S} Goals(\Theta)$;
(iv) $p(\Theta) = p^3(\Theta)$ for each $\Theta \in S$;
(v) $f(\delta) = f^3(\delta)$ for each $\delta \in \Delta$. □

4.2.6. Operational semantics

We are, at this stage, able to define our envisaged operational semantics of a temporal specification as the G-automaton synthesized by performing the steps described above. And note that the semantics proposed is not only correct with respect to the specification (in the sense that its induced life-structure satisfies the axioms) but it can also be proved to be representative, in a rather strong way, of the class of all G-automata satisfying the specification. It is canonical in the exact sense that, taking any other G-automaton satisfying the specification, its induced life-structure is a subset of the life-structure of its operational semantics.

Proposition 4.11. (Operational semantics)
Let syn be the G-automaton synthesized from a given specification $spec = \langle\Sigma, \Gamma\rangle$ with Σ a finite signature. Then:

(i) syn is goal-consistent and $syn \models spec$;
(ii) for each G-automaton aut over Σ, if $aut \models spec$ then $Lfs(aut) \subseteq Lfs(syn)$;
(iii) if syn has an empty set of states then $spec$ is inconsistent.

Proof. We present just an outline of the proof.

(i) By the construction in step 5 syn is clearly goal-consistent.
Consider now $\eta \in Comp_{syn}$, $i \in I\!N$ and $\varphi \in \Gamma$. It can be shown, by induction on the structure of φ, that $Lfc(\eta) \models_i \varphi$. We sketch the proof just for the base cases and for eventualities:

$\varphi \equiv b$, with $b \in \Sigma_{obs}$
If for some state $\Theta \in S$ it was the case that $b \notin S$ i.e. $b \notin v(S)$ then it would have to include $(\neg\, b)$. Due to step 1 that state would clearly have been rejected since it would include a contradiction. Therefore, by definition of $Lfc(\eta)$ it is clear that $b \in Lfc(\eta)_i$ and hence $Lfc(\eta) \models_i b$;

$\varphi \equiv \diamond a$, with $a \in \Sigma_{act}$
If for some state $\Theta \in S$ it was the case that $\diamond a \notin S$ then, necessarily, it would have to include $(\neg \diamond a)$. The previous argument applies to this case and hence the state would have been rejected in step 1. Moreover, since the state was maintained after step 3 it establishes its enablings and $\diamond a$ in particular. Therefore we have that $a \in Menu(\Theta)$. So, by definition of $Lfc(\eta)$ it is clear that $\diamond a \in Lfc(\eta)_i$ and hence $Lfc(\eta) \models_i \diamond a$;

$\varphi \equiv \triangledown a$, with $a \in \Sigma_{act}$
In this case, clearly, $S = \emptyset$ which makes the result trivial. Note that every state candidate is obviously eliminated in step 1 because the empty action contradicts $\triangledown a$;

$\varphi \equiv (\neg\, (\psi \;\mathsf{U}\; \vartheta))$
By construction, $dom(\eta_i)$ satisfies $(\neg\, \vartheta)$ and either $(\neg\, \psi)$ or $(\mathsf{X}((\neg\, (\psi \;\mathsf{U}\; \vartheta))))$. In the first case we are done. In the second case $(\neg\, \psi)$ is a target and gives rise to goal(s). Hence the acceptance of the computation ensures its fulfilment.

(ii) Let $\delta = \langle s, e, r\rangle$ be a transition of aut. In order to prove the desired property it is enough to guarantee the existence of a transition $\delta' = \langle s', e, r'\rangle$ of syn for which $v_{aut}(s) = v_{syn}(s')$, $v_{aut}(r) = v_{syn}(r')$, $Menu_{aut}(s) = Menu_{syn}(s')$ and $Menu_{aut}(r) = Menu_{syn}(r')$. For the purpose first consider the candidate state $\Theta^0 = \mathsf{G}(\Gamma) \cup \diamond Menu_{aut}(s) \cup obs(v_{aut}(s))$. If Θ^0 is eliminated in step 2 then pick Θ^1 such that $\langle \Theta^0, \emptyset, \Theta^1\rangle$ is obtained in step 1 by adding exactly the same observations and enablings. Repeatedly proceed in this manner until either some Θ^n is not eliminated or every possibility has been rejected (there are finitely many). Clearly this last case is not possible. It would mean that the state s of aut was incompatible with its mandatory stuttering transition. Hence, considering an appropriate branch Ω of $tableau(\Theta^n \cup \nabla(e))$ one gets the desired transition δ' precisely by choosing $s' = \Theta^n$ and $r' = next(\Omega) \cup \diamond(Menu_{aut}(r)) \cup obs(v_{aut}(r))$.

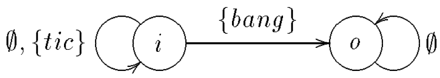
(iii) This an immediate consequence of the previous item. Clearly, no G-automaton satisfies the specification, since $Lfs(syn) = \emptyset$. □

The proof of the first statement of this proposition is a direct consequence of the tableaux constructions and of the essence behind the goal fulfilment condition in the acceptance condition of G-automata computations.

The second result can be obtained by inductively observing that it cannot be the case that any of the relevant states and transitions are thrown away in any of the eliminating steps of the synthesis algorithm.

Let us just examplify the synthesis method by applying it to two concrete specifications. We start by taking a look at the resulting operational semantics of the clock bomb system and then proceed to the G-automaton corresponding to the mutual exclusion protocol (both specified in subsection 2.2).

EXAMPLE 4.12. (Operational semantics of clock bomb)
The G-automaton synthesized from the $clock\_bomb$ specification in 2.9 is represented by the diagram below. Note that, of course, the goal is essential to the achievement of the liveness requirement expressed by axiom 5.



with:

$$i = \mathsf{G}(\Gamma_{clock\_bomb}) \cup \{(\mathsf{F}\,\nabla bang), on, \diamond tic, \diamond bang\}$$
$$o = \mathsf{G}(\Gamma_{clock\_bomb}) \cup \{(\neg\, on), (\neg \diamond tic), (\neg \diamond bang)\}$$

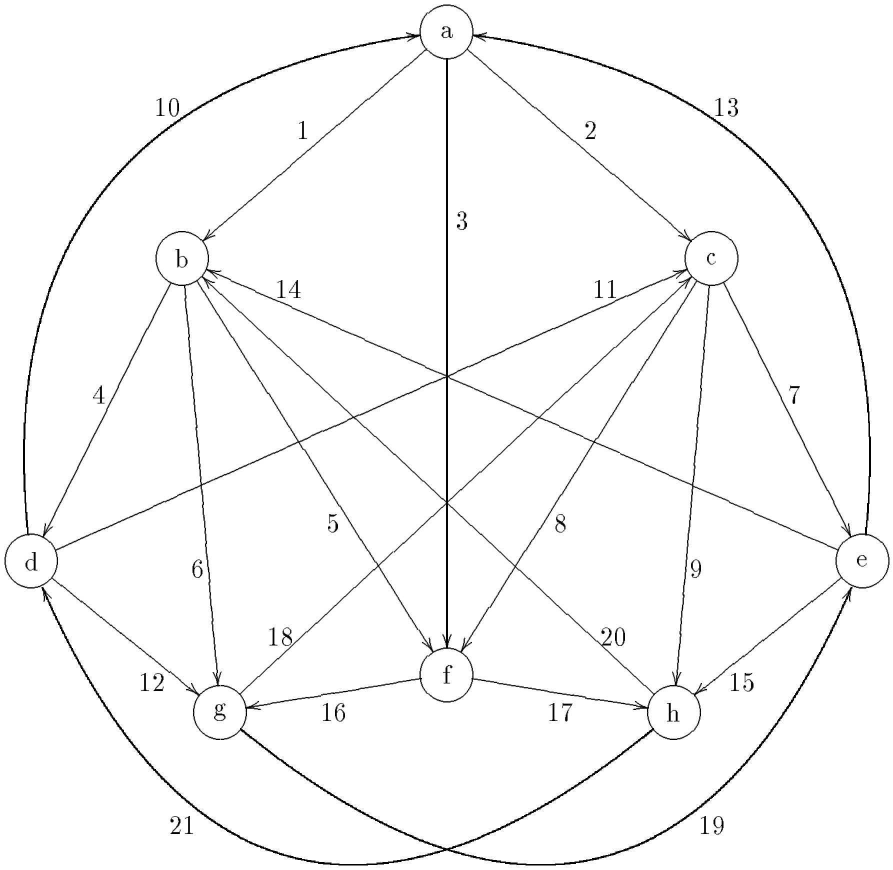
$v(i) = \{on\}$, $v(o) = \emptyset$

$G = \{\{\nabla bang\}\}$

$p(i) = \{\{\nabla bang\}\}$, $p(o) = \emptyset$

$f(i \stackrel{\{bang\}}{\to} o) = \{\{\nabla bang\}\}$, $f(i \stackrel{\emptyset}{\to} i) = f(i \stackrel{\{tic\}}{\to} i) = f(o \stackrel{\emptyset}{\to} o) = \emptyset$ □

EXAMPLE 4.13. (Operational semantics of mutual exclusion protocol)
The G-automaton synthesized from the *mutex* specification in 2.10 is represented by the diagram



where the stuttering transitions were omitted and the labels on transitions are described below:

Δ is the set containing the following transitions:

$\delta^1 = \langle a, \{willing_1\}, b\rangle$	$\delta^2 = \langle a, \{willing_2\}, c\rangle$
$\delta^3 = \langle a, \{willing_1, willing_2\}, f\rangle$	$\delta^4 = \langle b, \{in_1\}, d\rangle$
$\delta^5 = \langle b, \{willing_2\}, f\rangle$	$\delta^6 = \langle b, \{in_1, willing_2\}, g\rangle$
$\delta^7 = \langle c, \{in_2\}, e\rangle$	$\delta^8 = \langle c, \{willing_1\}, f\rangle$
$\delta^9 = \langle c, \{willing_1, in_2\}, h\rangle$	$\delta^{10} = \langle d, \{out_1\}, a\rangle$
$\delta^{11} = \langle d, \{out_1, willing_2\}, c\rangle$	$\delta^{12} = \langle d, \{willing_2\}, g\rangle$
$\delta^{13} = \langle e, \{out_2\}, a\rangle$	$\delta^{14} = \langle e, \{willing_1, out_2\}, b\rangle$
$\delta^{15} = \langle e, \{willing_1\}, h\rangle$	$\delta^{16} = \langle f, \{in_1\}, g\rangle$
$\delta^{17} = \langle f, \{in_2\}, h\rangle$	$\delta^{18} = \langle g, \{out_1\}, c\rangle$
$\delta^{19} = \langle g, \{out_1, in_2\}, e\rangle$	$\delta^{20} = \langle h, \{out_2\}, b\rangle$
$\delta^{21} = \langle h, \{in_1, out_2\}, d\rangle$	$\langle x, \emptyset, x\rangle$, for each state $x \in \{a, \ldots, h\}$

v is such that:

$$\begin{array}{ll} v(a) = \{idle_1, idle_2\} & v(b) = \{idle_2\} \\ v(c) = \{idle_1\} & v(d) = \{critical_1, idle_2\} \\ v(e) = \{idle_1, critical_2\} & v(f) = \emptyset \\ v(g) = \{critical_1\} & v(h) = \{critical_2\} \end{array}$$

$G = \{g_1, g_2\}$ with $g_1 = \{\triangledown out_1\}$ and $g_2 = \{\triangledown out_2\}$

p is such that:

$$\begin{array}{ll} p(a) = \emptyset & p(b) = \{g_1\} \\ p(c) = \{g_2\} & p(d) = \{g_1\} \\ p(e) = \{g_2\} & p(f) = \{g_1, g_2\} \\ p(g) = \{g_1, g_2\} & p(h) = \{g_1, g_2\} \end{array}$$

$f(\delta) = \emptyset$ except for

$$\begin{array}{l} f(\delta^{10}) = f(\delta^{11}) = f(\delta^{18}) = f(\delta^{19}) = \{g_1\} \\ f(\delta^{13}) = f(\delta^{14}) = f(\delta^{20}) = f(\delta^{21}) = \{g_2\} \end{array}$$

□

5. Executing G-automata

Obviously, any operational model is intended to capture some notion of "computation". And the conceptual breakthrough from automata to execution is necessarily thin because of their intrinsic resemblance to a state machine. In fact, a synthesis step such as ours really corresponds to "building an executable program from a non-executable specification" (Wolper, 1987). Therefore, we now turn our attention to the direct implementation of goal-consistent G-automata. If we manage to implement the acceptance condition of G-automata, and given that any specification over a finite signature can be thought of as being represented by a G-automaton (as shown in the previous section), any reasonable implementation of the synthesized G-automaton can be seen as a way of prototyping the specified system. Furthermore, by choosing such an approach to executing temporal logic specifications we do not have to assume any, a priori, strategy for implementing any desired fairness or liveness property. It is the way we choose to implement the acceptance condition of G-automata that must guarantee that all the specified properties are satisfied.

Moreover, since executing a specification is no more than building a model of its axioms (Fisher and Owens, 1995), and due to the correctness of the synthesized G-automaton with respect to the initial specification, any implementation strategy can be chosen provided that it is fair enough as to ensure that every pending goal at a certain step of the execution of a G-automaton is indeed fulfilled in some future time. Every possible life-cycle satisfying the given specification is represented in its synthesized G-automaton and therefore any of them may be the one resulting from execution, depending on the way the goal fulfilling strategy and also the inherent non-determinism of G-automata are implemented. This is the main difference with the respect, for instance, to the "declarative past, imperative future" paradigm of executing temporal logic specifications (Fisher and

Owens, 1992). In fact, the strategy followed by its application to METATEM (Barringer, Fisher, Gabbay, Gough and Owens, 1989) can obviously be seen as the search for a path in its corresponding automaton. But, of course, without knowing its complete structure, and even possibly having to unwind previous choices.

This happens because by executing G-automata we are not anymore concerned with consistency. The only remaining problem is the correct scheduling of goal fulfilment. The unpleasant situation when there is no model for the specification is immediately thrown away by the synthesis method. And there is also no way an intermediate decision during the execution step may lead to a dead end. All of these are clearly excluded from executing goal-consistent G-automata. Even if a performed transition did not help fulfilling a goal, we do always have a way of reaching its fulfilment by continuing its execution from the reached state. This avoids the use of backtracking strategies which is, of course, very convenient in the case of an interactive prototyping environment as suggested in (Fisher and Owens, 1992).

This main advantage is a direct consequence of a compiling instead of interpreting approach to programming languages. Consistency errors can be found at compilation time. While, of course, interpreting an inconsistent theory may lead to unexpected run time errors. The step of synthesis can therefore be seen as a compilation (as opposed to interpretation as in METATEM). The synthesized G-automaton plays the role of the abstract machine code generated by the compilation. And note that several runs of the same system can then be tried without further analysis of the specification since the G-automaton is synthesized once and for all. Another usual advantage of compiling resides in the possibility of composing programs. This will be briefly discussed in the conclusions, namely with respect to parallel composition.

But compiling also has its disadvantages. Namely because it is clear that interpretation is much more convenient for incremental specification and debugging.

Now returning to the central problem of implementing G-automata, the "unique" difference with respect to the implementation of classical automata is clearly the additional requirement on their acceptance condition. In other terms, we face the problem of choosing a general strategy for implementing eventualities. Of course, we may wonder if such a general method exists. We could simply transmit a report of the current pending goals and the menu of possible transitions and expect a human to take a wise choice. This would not, however, guarantee acceptance. On the other hand, it is easy to see that it will suffice to queue pending goals and always choose a transition which may help to fulfil the oldest goal in the queue. Since the G-automata we are interested in are finite and goal-consistent this can be shown to be enough. In what concerns possible non-determinism we may assume a randomized choice is taken.

We should not, nevertheless, neglect the possibility of prototyping the specified systems in an interactive environment. I.e., it would be desirable to allow external signals on actions have their counterpart in terms of the strategy for choosing the transition to be executed. With an approach such as ours, this can be easily done, without further constraints, by restricting the choice in each step to transitions whose occurrence maximizes the set of actions signaled by the environment. The choice would therefore be only among those actions which, besides being chosen in order to contribute to the fulfilment of the oldest pending goal (if any), ensure the occurrence of the greatest possible number of signaled actions. By doing so, of course, external signal on actions my have to wait for their execution.

Obviously, the case when the set of states of the G-automaton is empty need not be considered. As far as synthesized automata are concerned, proposition 4.11 guarantees that this only happens in case the given specification is inconsistent.

Definition 5.1. (Interactive execution of G-automata)
Let $aut = (\Sigma, S, \Delta, v, G, p, f)$ be a finite goal-consistent G-automata with $S \neq \emptyset$. The strategy for implementing aut is:

1 choose $s_0 \in S$;
2 make $i = 0$;
3 initialize *queue* to contain (in some order) the set $p(s_0)$;
4 loop

> read the current set of external action stimuli to e;
> if $queue = empty$ then
>
> – choose $\eta_i \in \Delta(s_i)$ maximizing $act(\eta_i) \cap e$;
> – make $s_{i+1} = cod(\eta_i)$
>
> else
>
> – choose $\eta_i \in \Delta(s_i)$ with $first(queue) \in Contrib(\eta_i)$ maximizing $act(\eta_i) \cap e$;
> – make $s_{i+1} = cod(\eta_i)$;
> – remove $f(\eta_i)$ from *queue* (keeping the order of the remaining goals)
>
> insert $p(s_{i+1}) \setminus (p(s_i) \setminus f(\eta_i))$ into *queue* (in some order);
> signal the execution of $act(\eta_i) \cap e$;
> increment i

end loop. □

Proposition 5.2. (Correctness of implementation)
Let $\eta : I\!\!N \rightarrow \Delta$ be the sequence of transitions generated by interactively executing a finite goal-consistent G-automaton aut with a non-empty set of states. Then, η is a computation of aut. □

The result is immediate since any goal in the head of *queue* is indeed fulfilled (the existence of a suitable transition is ensured by the fact that the G-automaton is goal-consistent and by the inner-cycle absence condition in the definition of contribution). Even in a state with no pending goals the open-nature conditions guarantees the existence of at least one transition leaving the state. All the rest is a consequence of the order imposed by the queueing mechanism and is of course independent of the randomized choice being used. Furthermore, in each step, the chosen transition executes as much externally signaled actions as possible.

Note that, in order to execute a specification, the initial state may be chosen according to a given input set of state constraints (eg. a condition on the values of observations, provided it is consistent with the specification).

PROPOSITION 5.3. (Interactive execution of temporal specifications)
Let $spec = \langle \Sigma, \Gamma \rangle$ be a specification over a finite signature and assume that syn, the G-automaton synthesized from $spec$, has a non-empty set of states. Then, $\eta : I\!N \rightarrow \Delta$, the sequence of transitions of syn generated by its interactive execution is such that $Lfc(\eta)$ satisfies $spec$. □

This result is clearly a corollary of both the correctness of the operational semantics and the previous proposition.

Let us now take a close look at two meaningful examples. Precisely the clock bomb system and the mutual exclusion protocol previously presented.

EXAMPLE 5.4. (Executing *clock_bomb*)
Recall the specification of the clock bomb system 2.9 and its operational semantics 4.12. Clearly, the only interesting initial state choice is i (eg. by requiring the constraint on). According to the outlined strategy, the unique goal $\{\triangledown bang\}$ is queued. Thus, immediately in the first step of the loop, the only possible transition contributing to the fulfilment of the goal is exactly $\langle i, \{\triangledown bang\}, o\rangle$. So, life-cycle λ obtained from the execution is such that:

$$\lambda_0 = \{on, \diamond tic, \diamond bang, \triangledown bang\},$$
$$\lambda_n = \emptyset \text{ for every } n > 0.$$ □

The strategy was clearly too strict and, of course, the non-determinism on the occurrence of $bang$ was merely implemented as if it had been specified $(\mathsf{X}\,\triangledown bang)$. And note that external signals requiring the occurrence of tic would not help. But, of course, the obtained life-cycle matches the specification.

EXAMPLE 5.5. (Executing *mutex*)
Recall the specification of the mutual exclusion protocol 2.10 and its operational semantics 4.13. For the sake of simplicity let us assume that the initial state choice is a (eg. by requiring $(idle_1 \wedge idle_2)$). According to the outlined strategy, since there are no pending goals, it is possible, for instance, that always the stuttering transition on state a is performed. Let us suppose, however, that there were always persistent external stimuli for both $willing_1$ and $willing_2$. In that case, clearly, the first transition chosen leads to state f. Therefore, both g_1 and g_2 become pending. Without loss of generality let us assume priority is given to g_1. Clearly, the transition contributing to achieving g_1 is δ^{16}. We are then at state g and for fulfilling g_1 two choices can be taken. Both moving to states c (by δ^{18}) and e (by δ^{19}) can be considered. Let us now suppose δ^{19} is taken. The goal g_1 is fulfilled and process 2 imediately enters the CS. The subsequent step is therefore fulfilling g_2 by moving to b by transition δ^{14} hence responding to the external signal for $willing_1$. After that, again state g is visited and so forth. In this case, the corresponding life-cycle λ is such that:

$$\lambda_0 = \{idle_1, idle_2, \diamond willing_1, \diamond willing_2, \triangledown willing_1, \triangledown willing_2\},$$
$$\lambda_1 = \{\diamond in_1, \diamond in_2, \triangledown in_1\},$$
$$\lambda_{2+3n} = \{critical_1, \diamond out_1, \diamond in_2, \triangledown out_1, \triangledown in_2\},$$

$$\lambda_{3+3n} = \{idle_1, critical_2, \diamond willing_1, \diamond out_2, \triangledown willing_1, \triangledown out_2\},$$
$$\lambda_{4+3n} = \{idle_2, \diamond in_1, \diamond willing_2, \triangledown in_1, \triangledown willing_2\}, \text{ for } n \in I\!N. \qquad \square$$

Clearly, the strategy schedules the two processes correctly. Of course the order is maintained, always the first process is chosen, but only because we assumed that was the non-deterministic choice.

Note that eventualities may reflect two kinds of intentions from the specifier view point. Either guaranteeing response requirements or merely requiring non-determinism on the moment of fulfilment (implicit versus intended non-determinism). The clock bomb example is clearly in the latest case. The eventuality is used as a means of underspecification since the system specified is not fully known. And obviously the computation given by executing it is for sure not the most pleasant. Similar problems arise, however, in other approaches to executing temporal logic programs (Abadi and Manna, 1989), (Fisher and Owens, 1992).

6. Conclusion and outlook

We have shown how to synthesize a representative canonical G-automaton satisfying a temporal logic specification. Namely, we have managed to derive liveness goals from a specification, hence being able to go beyond simply dealing with safety properties. Having obtained a way of associating an operational counterpart, in a suitable domain, to each system specification, several extensions to this work can now be considered. First of all, a full study of the compositionality properties of G-automata and of the connection between these constructs and the specification logic is required. The obvious step should be the study of the associated institution and its relationship to the corresponding denotational semantics (Sernadas, Sernadas and Caleiro, 1996). A first attempt in this direction can be found in (Caleiro, 1995).

A deeper study of the power and nature of the acceptance condition of G-automata is necessary. However, we are already able to compare it with the classical Muller automata on infinite objects (Thomas, 1990). Büchi automata are not suitable to our purposes since their acceptance condition is based on passing infinitely often in one of a number of final states, which does not fit our need for stuttering (the persistent performance of the stuttering transition at an accepting state would be accepted but could, in general, be unaware of goal fulfilment). With respect to Muller automata we can say that G-automata are not less powerful i.e., that a G-automaton can be build out of a Muller automaton both accepting exactly the same computations. Muller automata acceptance condition is that the set of states visited infinitely often is one of a number of sets of states (called frequent sets). This is done by replication of the original state/transition structure through each frequent set and introduction of a persistent goal, in each of the replications, corresponding to each of its states, whose fulfilment is attained exactly by the transitions reaching that state. The acceptance condition of G-automata seems, however, more intuitive since it has to do with transitions rather than with states. At each step, each transition performed may be “helping” the computation to be accepted. This seems much more natural and feasible than being sure that in the end an exact set of states was visited infinitely often.

It is clear that the G-automata framework is suitable as a means of implementing prototypes of temporal systems specifications. However, the difficulties remain on the goal structure of the automaton. Several possibilities can be considered concerning the way eventualities are fulfilled. The study and implementation of the strategy proposed and others are a further possible extension of the present work. An interesting line of reserach would be considering some kind of probabilistic automata (Pnueli, 1983), (Segala, 1995). By doing that we also make possible a probabilistic study of the correctness of implementation strategies and open the door to the use of probabilistic temporal logic (Hart and Sharir, 1984).

Another important subject is the study of compositionality results. The fact that a system specification can be compiled into a G-automaton may be very interestingly used together with, for instance, a parallel composition primitive in order to implement interconnections of concurrent systems. The framework can therefore be rather compositional and therefore used for prototyping concurrent systems (Fisher and Barringer, 1991), (Merz, 1992), (Fisher and Wooldridge, 1993).

A severe drawback is that the ideas presented cannot be fully applied to first-order temporal logic (since it is incomplete). But, as pointed out by several authors, propositional temporal logic is enough in many concrete applications (Manna and Wolper, 1984), (Emerson, 1990).

A direct application of the presented techniques is their use for monitoring systems with respect to a given temporal specification. Monitoring of temporal logic constraints is a way to ensure dynamic integrity for example in the database area (Chomicki, 1992), (Kung, 1984), (Lipeck, 1990), (Lipeck, Gertz and Saake, 1994), (Lipeck and Saake, 1987) or in object systems (Schwiderski, Hartmann and Saake, 1994). In (Hülsmann and Saake, 1991), (Lipeck and Saake, 1987), (Saake, 1991), (Schwiderski, Hartmann and Saake, 1994) a special kind of automaton, called transition graph, is constructed from temporal logic constraints to be used for monitoring dynamic integrity in databases. Compared to G-automata, transition graphs take no advantage of explicit goals such only distinguishing the situations "no pending goals", "pending goals which are satisfiable" and "inconsistent set of pending goals". Furthermore, satisfaction of the specification can only be defined for finite system runs. Compared to this work, G-automata enable a differentiated measure of the satisfaction of a temporal specification because the goals are made explicit. We can, for instance, use a preference relation between goal sets to distinguish system states which are "better" in satisfying a certain specification than others. During runtime, the process of system monitoring enables decisions about plans for satisfying more goals depending on the current state. This also allows for coordination of systems behaviour based on a temporal specification without having to generate the complete system from scratch.

References

M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

J. Adámek, H. Herrlich, and G. Strecker (1990). *Abstract and concrete categories*. John Wiley.

H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: a framework for programming in temporal logic. In *Proceedings of REX workshop on stepwise refinement of distributed systems: models, formalisms, correctness - LNCS 430*. Springer Verlag, 1989.

H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal specifications. In *Proceedings of the 16th ACM Symposium on Theory of Computing*, pages 51–63, 1984.

J. Bell and M. Machover. *A Course in Mathematical Logic*. North-Holland, 1977.

M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, pages 207–226, 1983.

C. Caleiro. Operational semantics of temporal object specifications. *The INESC Journal on Junior Activities in Science and Technology, 1(1)*, pages 7–35, 1994.

C. Caleiro. On the relationship between operational and denotational semantics of temporal logic specification of object behaviour. In R. Wieringa and R. Feenstra, editors, *Information systems correctness and reusability. Selected papers from the IS-CORE Workshop, Amsterdam 94*, pages 69–83. World Scientific, 1995.

J. Chomicki. History-less checking of dynamic integrity constraints. In F. Golshani, editor, *Proceedings of the 8th IEEE Conference on Data Engineering*, pages 557–564, 1992.

E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Workshop on Logics of Programs - LNCS 131*, pages 52–71, 1981.

E. Clarke, O. Grumberg, and R. Kurshan. A synthesis of two approaches for verifying finite state concurrent systems. *Journal of Logic and Computation, 2(5)*, pages 605–618, 1992.

E. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, pages 996–1072. 1990.

J. Fiadeiro and T. Maibaum. Temporal theories as modularization units for concurrent systems specification. *Formal Aspects of Computing*, 4:239–272, 1992.

M. Fisher and H. Barringer. Concurrent METATEM processes - a language for distributed AI. In *Proceedings of the European Simulation Multiconference*, 1991.

M. Fisher and R. Owens. From the past to the future: executing temporal logic programs. In *Proceedings of logic programming and automated reasoning - LNCS 624*. Springer Verlag, 1992.

M. Fisher and R. Owens. Executable modal and temporal logics - a survey. In *Executable modal and temporal logics - LNAI 897*. Springer Verlag, 1995.

M. Fisher and M. Wooldridge. Executable temporal logic for distributed AI. In *Twelfth International Conference on Distributed AI*. 1987.

N. Francez. *Fairness*. Springer-Verlag, 1986.

D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, 1980.

R. Goldblatt. *Logics of Time and Computation*. CSLI, 1987.

S. Hart and M. Sharir. Probabilistic temporal logics for finite and bounded models. In *Proceedings of the 16th ACM Symposium on Theory of Computing*, pages 1–13, 1984.

K. Hülsmann and G. Saake. Theoretical foundations of handling large substitution sets in temporal integrity monitoring. *Acta Informatica*, 28:365–407, 1991.

C. Kung. Temporal framework for database specification and verification. In *Proceedings of the 5th International Conference on Very Large Databases*, pages 91–99, 1984.

U. Lipeck. Transformation of dynamic integrity constraints into transaction specifications. *Theoretical Computer Science*, 76:115–142, 1990.

U. Lipeck, M. Gertz, and G. Saake. Transitional monitoring of dynamic integrity constraints. *Data Engineering*, 17(2):38–42, 1994.

U. Lipeck and G. Saake. Monitoring dynamic integrity constraints based on temporal logic. *Information Systems*, 12:255–269, 1987.

Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *Temporal Logic in Specification - LNCS 398*, pages 124–164. Springer Verlag, 1987.

Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear time, branching time and partial order in logics and models for concurrency - LNCS 354*, pages 201–284. Springer-Verlag, 1989.

Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83:97–130, 1991.

Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

Z. Manna and A. Pnueli. A temporal proof methodology for reactive systems. In *Program design calculi - NATO ASI Series, Series F: Computer and System Sciences*. Springer-Verlag, 1993.

Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems, 6(1)*, pages 68–93, 1984.

S. Merz. *Temporal logic as a programming language*. PhD thesis, Ludwig-Maximilians Universität, München, Germany, 1992.

A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

A. Pnueli. On the extremely fair termination of probabilistic programs. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 278–290, 1983.

A. Pnueli. System specification and refinement in temporal logic. In *Foundations of Software Technology and Theoretical Computer Science, LNCS 652*, pages 1–38. Springer-Verlag, 1992.

A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, pages 179–190, 1988.

G. Saake. Descriptive specification of database object behaviour. *Data and Knowledge Engineering*, 6(1):47–73, 1991.

S. Schwiderski, T. Hartmann, and G. Saake. Monitoring temporal preconditions in a behaviour oriented object model. *Data and Knowledge Engineering*, 14:143–186, 1994.

R. Segala. A compositional trace-based semantics for probabilistic automata. In *Proceedings of CONCUR95 - LNCS 962*, pages 234–248. Springer Verlag, 1995.

A. Sernadas. Temporal aspects of logical procedure definition. *Information Systems*, 5:167–187, 1980.

A. Sernadas, J. F. Costa, and C. Sernadas. An institution of object behaviour. In H. Ehrig and F. Orejas, editors, *Recent trends in data type specification: 9th workshop on specification of abstract data types - Selected papers*, pages 337–350. Springer Verlag, 1994.

A. Sernadas, C. Sernadas, and C. Caleiro. Denotational semantics of object specification. Technical report, IST, Lisbon, Portugal, 1996. Submitted for publication.

A. Sernadas, C. Sernadas, and J. F. Costa. Object specification logic. *Journal of Logic and Computation*, 5(5):603–630, 1995.

A. Sernadas, C. Sernadas, and J. Ramos. A temporal logic approach to object certification. *Journal of Data and Knowledge Engineering*, 1996. In print.

W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, pages 133–191. 1990.

P. Wolper. On the relation of programs and computations to models of temporal logic. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification - LNCS 398*, pages 75–123. 1987.