Goal-Driven Operational Semantics of Temporal Specification of Reactive Systems Behaviour[§]

Carlos Caleiro[†]
Gunter Saake[‡]
Amílcar Sernadas[†]

Abstract

A propositional linear 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. Applications to monitoring are discussed.

Keywords: reactive system, temporal logic specification, tableau, eventuality, goal, G-automaton, 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 [2, 8, 10, 22, 24, 26, 32, 33] and in monitoring [14, 15, 18, 29]. The advantages are known to lie on the clear declarative formalisation 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.

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 [30] and first dealt with in [5], 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 [31] concerning denotational semantics also stressed the need for providing an adequate

[†]Departamento de Matemática, Instituto Superior Técnico, Av. Rovisco Pais, 1000 Lisboa, Portugal, email: {ccal|acs}@inesc.pt

[‡]Institut für Technische Informationssysteme, Universität Magdeburg, Universitätsplatz, D-39016 Magdeburg, Germany, email: saake@iti.cs.uni-magdeburg.de

[§]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**).

operational counterpart. For the purpose we present G-automata, which are labelled transition systems with stuttering, additionally equipped with a structure of goals whose intention is to restrict acceptance. This is closely related to the "fairness" assumptions on the behaviour of the system [22] and somehow similar to the acceptance structure of other, well known, definitions of automata on infinite words such as Büchi and Muller automata [34] or $\forall$-automata [19]. However, G-automata arise, to our view, much more naturally, while still emulating their expressible power.

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 specified through temporal logic by means of a tableaux-like method based on the decision procedure for temporal logic satisfiability described in [4, 25]. Note that fairness or liveness properties can be specified, having an operational counterpart reflected on the acceptance condition of the obtained automaton. We point out how the results can be useful in monitoring applications and compare with related work.

Through the paper, we assume that the reader is conversant with the field of temporal logic specification, for instance at the level of [9, 13, 27], and with fairness and liveness issues [11, 12].

In section 2 we introduce and illustrate the use of linear 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 closer 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 discusses the results obtained, their connection with previous work and applications. We also outlook 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 occurrences some local state is reached and may be observable 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 [30], we consider a system specification to be a description of its actions and observations as well as a characterisation of the relationship between distinct actions and observations, of the enabling conditions of actions and of the effects of each occurrence on observations. In general, 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) [33], is a propositional linear temporal logic, with the possibility of expressing state observations and which distinguishes between action occurrence and action enabling. So, if a is some action (or prime action, as we shall name it), ∇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. $\square$

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 $Atom_\Sigma = \{b : b \in \Sigma_{obs}\} \cup \{\nabla a : a \in \Sigma_{act}\} \cup \{\diamond a : a \in \Sigma_{act}\}$. $\square$

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

- $Atom_\Sigma \subseteq Form_\Sigma$;
- $(\neg\, \varphi) \in Form_\Sigma$ provided that $\varphi \in Form_\Sigma$;
- $(\varphi \Rightarrow \psi) \in Form_\Sigma$ provided that $\varphi, \psi \in Form_\Sigma$;
- $(\mathsf{X}\, \varphi) \in Form_\Sigma$ provided that $\varphi \in Form_\Sigma$;
- $(\varphi\, \mathsf{U}\, \psi) \in Form_\Sigma$ provided that $\varphi, \psi \in Form_\Sigma$. $\square$

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

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

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

Satisfaction is, as usual, defined over Kripke-like structures to which we call *life-cycles*. A life-cycle consists of a sequence of sets of observations and enabled and occurring prime actions, where every occurring prime action must be enabled. To such sets we call *snapshots*.

Definition 2.4 (Snapshots and life-cycles)
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. A *life-cycle* over Σ is a map $\lambda : I\!N \rightarrow Snp_\Sigma$. $\square$

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

- $\lambda \models_i b$ iff $b \in \lambda_i$, provided that $b \in \Sigma_{obs}$;
- $\lambda \models_i \diamond a$ iff $\diamond a \in \lambda_i$, provided that $a \in \Sigma_{act}$;
- $\lambda \models_i \triangledown a$ iff $\triangledown a \in \lambda_i$, provided that $a \in \Sigma_{act}$;
- $\lambda \models_i (\neg\, \varphi)$ 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 $\varphi \in \Phi$. $\square$

The temporal logic we use is non-anchored [20] and is not equipped with a special (initial) atom whose satisfaction is only held at 0. This happens because we are interested in specifying overall properties of systems behaviour with no particular interest in initial constraints.

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-cycle λ over Σ satisfies *spec*, $\lambda \models spec$, iff $\lambda \models \Gamma$. $\square$

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 shall also be intended, later on, that the meaning of $\diamond a$ is exactly that prime action a can occur (that is, enabling of a prime action could be considered as an observation whose value in each point reflected its possible occurrence). We also require that the systems we specify are open towards the environment. By this we mean that occurrences may be finitely delayed, in any of its life-cycles, in order to allow interleaving with life-cycles of other systems. This shall mean that formulae of the form $\triangledown a$ cannot be expected to hold.

We start by taking a brief look at some commonly used classes of formulae for specifying systems properties.

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

Studies of temporal logic specifiable properties can be found, for instance, in [20, 21].

Example 2.8 (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 assures 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}:

- λ such that $\lambda_i = \{on, \diamond tic, \diamond bang, \triangledown tic\}$ for every $i \in I\!N$;
- λ' such that $\lambda'_0 = \{on, \diamond tic, \diamond bang, \triangledown tic\}$, $\lambda'_1 = \{on, \diamond tic, \diamond bang, \triangledown bang\}$, $\lambda'_i = \emptyset$ for every $i > 1$;
- λ'' such that
- $\lambda''_i = \{on, \diamond tic, \diamond bang, \triangledown tic\}$ for every $i < 50$, $\lambda''_{50} = \{on, \diamond tic, \diamond bang, \triangledown bang\}$, $\lambda''_i = \emptyset$ for every $i > 50$.

Clearly, λ' and λ'' satisfy $clock\_bomb$ but λ does not satisfy axiom 6. □

[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 $i \in I\!N$.

3 Which operational semantics?

If we should adopt a class of labelled transition systems whose infinite computations were extracted, as usual, by just requiring sequentiality, we would have no natural way of restricting its behaviour in a way which would support for instance the satisfaction of fairness properties. In the field of systems specification this is often done by imposing very strong restrictions to the occurrence of actions [22, 23]. However, this may not be wanted. One may want to specify systems where, to some of its actions no fairness is required or required only on certain situations (for instance, when some observation is true). This apparent freedom copes well with compositionality and with the need for a mandatory stuttering transition. The treatment of fairness may, if so specified, allow the exclusion of computations where a stuttering transition is persistently performed while other actions are enabled. The operational domain we are looking for must also be such that the exact behaviour of a specified system may be synthesised (finitely) from the specification. Another important issue concerns the meaning of enabling. A state should have an enabled action only if that action takes part in a possible transition from that state.

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.

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

- a *signature* Σ of observation and prime action symbols;
- a set S of *states*;
- a set $\Delta \subseteq S \otimes 2^{\Sigma_{act}} \otimes S$ 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 leading from state s to state r;
- a *state valuation map* $v : S \to 2^{\Sigma_{obs}}$, which assigns to each state its set of satisfied observations;
- a set G of *goals*;
- a *pending goals map* $p : S \to 2^G$, which defines the set of pending goals at each state;
- a *fulfilled goals map* $f : \Delta \to 2^G$, which maps each transition onto the set of goals it actually 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 with 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 synchronise, 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 contribute to the eventual achievement of a pending goal. The intuition is that the transition contributes to its achievement 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 $i \in \{1, ..., n-1\}$;

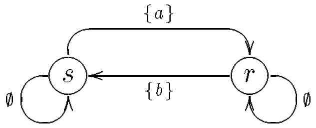
- *initialisation*: $\delta_1 = \delta$;

- *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 is cyclic);
- $Contrib(r \xrightarrow{\emptyset} r) = \emptyset$ (every sequence is cyclic);
- $Contrib(s \xrightarrow{\{a\}} r) = \{g'\}$ (consider $s \xrightarrow{\{a\}} r \xrightarrow{\{b\}} s$);
- $Contrib(r \xrightarrow{\{b\}} s) = \{g'\}$ (consider $r \xrightarrow{\{b\}} s$). $\square$

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 [5, 22, 30].

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 $\langle s, e, r\rangle \in \Delta$. $\square$

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.

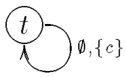
To illustrate some of the potentialities of this semantic domain, we shall pay closer attention to its ability to support simple parallel composition with preservation of the goal mechanisms of the parts involved.

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:

- $\Sigma = (\Sigma'_{obs} \oplus \Sigma''_{obs}, \Sigma'_{act} \oplus \Sigma''_{act})$;
- $S = S' \otimes S''$;
- $\Delta = \{\langle\langle s', s''\rangle, e' \oplus e'', \langle r', r''\rangle\rangle : \langle s', e', r'\rangle \in \Delta', \langle s'', e'', r''\rangle \in \Delta''\}$;
- $v(\langle s', s''\rangle = v'(s') \oplus v''(s'')$ for every $s' \in S'$ and $s'' \in S''$;
- $p(\langle s', s''\rangle = p'(s') \oplus p''(s'')$ for every $s' \in S'$ and $s'' \in S''$;
- $\phi(\langle\langle s', s''\rangle, e' \oplus e'', \langle r', r''\rangle\rangle) = f'(\langle s', e', r'\rangle) \oplus f''(\langle s'', e'', r''\rangle)$ for every $(s', e', r') \in \Delta'$ and $(s'', e'', r'') \in \Delta''$. □

Note that $\oplus$ stands for disjoint union and $\otimes$ for cartesian product. Moreover, parallel composition is a product in the corresponding category[2] of G-automata [6].

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

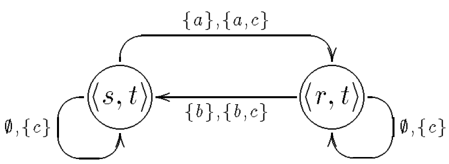


and with:

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

[2] Refer to [1] for a textbook on category theory.

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



and with:

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

As was motivated before, the introduction of goals into transition systems was brought up as a way of restricting the usual notion of computation, imposing necessary fulfilment of every pending goal at each ongoing state. Due to this fact we shall restrict our attention to goal-consistent G-automata i.e., G-automata where every pending goal at a certain state 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* if the following condition is satisfied:

- 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, 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 this notion of consistency is preserved through parallel composition.

Theorem 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 undesired computations 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 $\eta{:}I\!N \to \Delta$ such that:

- $dom(\eta_{i+1}) = cod(\eta_i)$ for every $i \in I\!N$;
- for each $i \in I\!N$, 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 closely related to the above defined satisfaction by life-cycle. As must be obvious, a life-cycle can be extracted from each computation of an automaton. Thus, we demand that the life-cycle corresponding to each computation of the automaton satisfies the formula.

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 all prime actions whose occurrence is enabled (possible) when 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 of snapshots $Lfc(\eta)$ such that $Lfc(\eta)_i = v(dom(\eta_i)) \cup \{\diamond a : a \in Menu(dom(\eta_i))\} \cup \{\nabla a : a \in act(\eta_i)\}$ 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:

- η s.t. $\eta_i = \langle s, \emptyset, s\rangle$ for every $i \in I\!N$ is not a computation of aut';
- η' s.t. $\eta'_i = \langle s, \{a\}, r\rangle$ for even i and $\eta'_i = \langle r, \{b\}, s\rangle$ for odd i is a computation of aut';
- $Lfc(\eta')_i = \{\diamond a, \nabla a\}$ for even i and $Lfc(\eta')_i = \{x, \diamond b, \nabla b\}$ for odd i. □

Definition 3.14 (Satisfaction by G-automaton)
We say that aut satisfies $\varphi \in Form_\Sigma$, $aut \models \varphi$, iff $Lfc(\eta) \models \varphi$ for each $\eta \in Comp_{aut}$. 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 Synthesising G-automata from specifications

The aim of this section is to present 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. The same is to say that the synthesised G-automaton must be in some sense canonical among all those G-automata which satisfy the specification. The synthesis algorithm here presented capitalises on the results obtained in [5] and consists of an adapted and extended version of the tableaux-based decision procedure for temporal logic presented in [4] and, more specifically, of the synthesis method proposed in [25]. We start by introducing the envisaged tableaux system [3] for the temporal logic we are using.

4.1 Tableaux for temporal logic

A tableau is a finite tree to whose nodes are associated sets of formulae. Given a certain set of tree formation (inference) rules and a set of formulae we can build up a tableau whose root node is the given set of formulae. The rules we are interested in, which we describe below, are similar to those used both in [4, 25]. The idea behind them is that all the reasoning at one instant in time can be resolved by making suitable requirements over the next instant.

In order to present the rules, we consider fixed a signature Σ.

Definition 4.1 (Inference rules schemata, inference rules, premises, conclusion)
Let $\varphi,\psi \in Form_\Sigma$. The *inference rules schemata* over Σ are:

- $(\neg\,\neg)$ *rule*
  $\varphi \therefore (\neg\,(\neg\,\varphi))$
- $(\Rightarrow)$ *rule*
  $(\neg\,\varphi)\ ;\ \psi \therefore (\varphi \Rightarrow \psi)$
- $(\neg \Rightarrow)$ *rule*
  $\varphi,(\neg\,\psi) \therefore (\neg\,(\varphi \Rightarrow \psi))$
- $(\neg\,\mathsf{X})$ *rule*
  $(\mathsf{X}\,(\neg\,\varphi)) \therefore (\neg\,(\mathsf{X}\,\varphi))$
- (U) *rule*
  $\psi\ ;\ \varphi,(\mathsf{X}\,(\varphi\,\mathsf{U}\,\psi)) \therefore (\varphi\,\mathsf{U}\,\psi)$
- $(\neg\,\mathsf{U})$ *rule*
  $(\neg\,\varphi),(\neg\,\psi)\ ;\ (\mathsf{X}\,(\neg\,(\varphi\,\mathsf{U}\,\psi))),(\neg\,\psi) \therefore (\neg\,(\varphi\,\mathsf{U}\,\psi))$.

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

If $\theta_{11},\ldots,\theta_{1k_1},\ldots,\theta_{n1},\ldots,\theta_{nk_n}$ are formulae and $\theta_{11},\ldots,\theta_{1k_1}\ ;\ \ldots\ ;\ \theta_{n1},\ldots,\theta_{nk_n} \therefore \theta$ is an inference rule we say that $\theta_{i1},\ldots,\theta_{ik_i}$ is a premise of the rule, for each $i \in \{1,\ldots,n\}$, and that θ is the conclusion of the rule. □

Clearly, 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:

- $(\vee)$ *rule*
  $\varphi\ ;\ \psi\ \therefore (\varphi \vee \psi)$
- $(\neg \vee)$ *rule*
  $(\neg\, \varphi), (\neg\, \psi) \therefore (\neg(\varphi \vee \psi))$
- $(\wedge)$ *rule*
  $\varphi, \psi \therefore (\varphi \wedge \psi)$
- $(\neg \wedge)$ *rule*
  $(\neg\, \varphi)\ ;\ (\neg\, \psi) \therefore (\neg(\varphi \wedge \psi))$
- $(\Leftrightarrow)$ *rule*
  $\varphi, \psi\ ;\ (\neg\, \varphi), (\neg\, \psi) \therefore (\varphi \Leftrightarrow \psi)$
- $(\neg \Leftrightarrow)$ *rule*
  $\varphi, (\neg\, \psi)\ ;\ (\neg\, \varphi), \psi \therefore (\neg(\varphi \Leftrightarrow \psi))$
- (G) *rule*
  $\varphi, (\mathsf{X}(\mathsf{G}\, \varphi)) \therefore (\mathsf{G}\, \varphi)$
- $(\neg\, \mathsf{G})$ *rule*
  $(\neg\, \varphi)\ ;\ (\mathsf{X}(\neg(\mathsf{G}\, \varphi))) \therefore (\neg(\mathsf{G}\, \varphi))$
- (F) *rule*
  $\varphi\ ;\ (\mathsf{X}(\mathsf{F}\, \varphi)) \therefore (\mathsf{F}\, \varphi)$
- $(\neg\, \mathsf{F})$ *rule*
  $(\neg\, \varphi), (\mathsf{X}(\neg(\mathsf{F}\, \varphi))) \therefore (\neg(\mathsf{F}\, \varphi))$

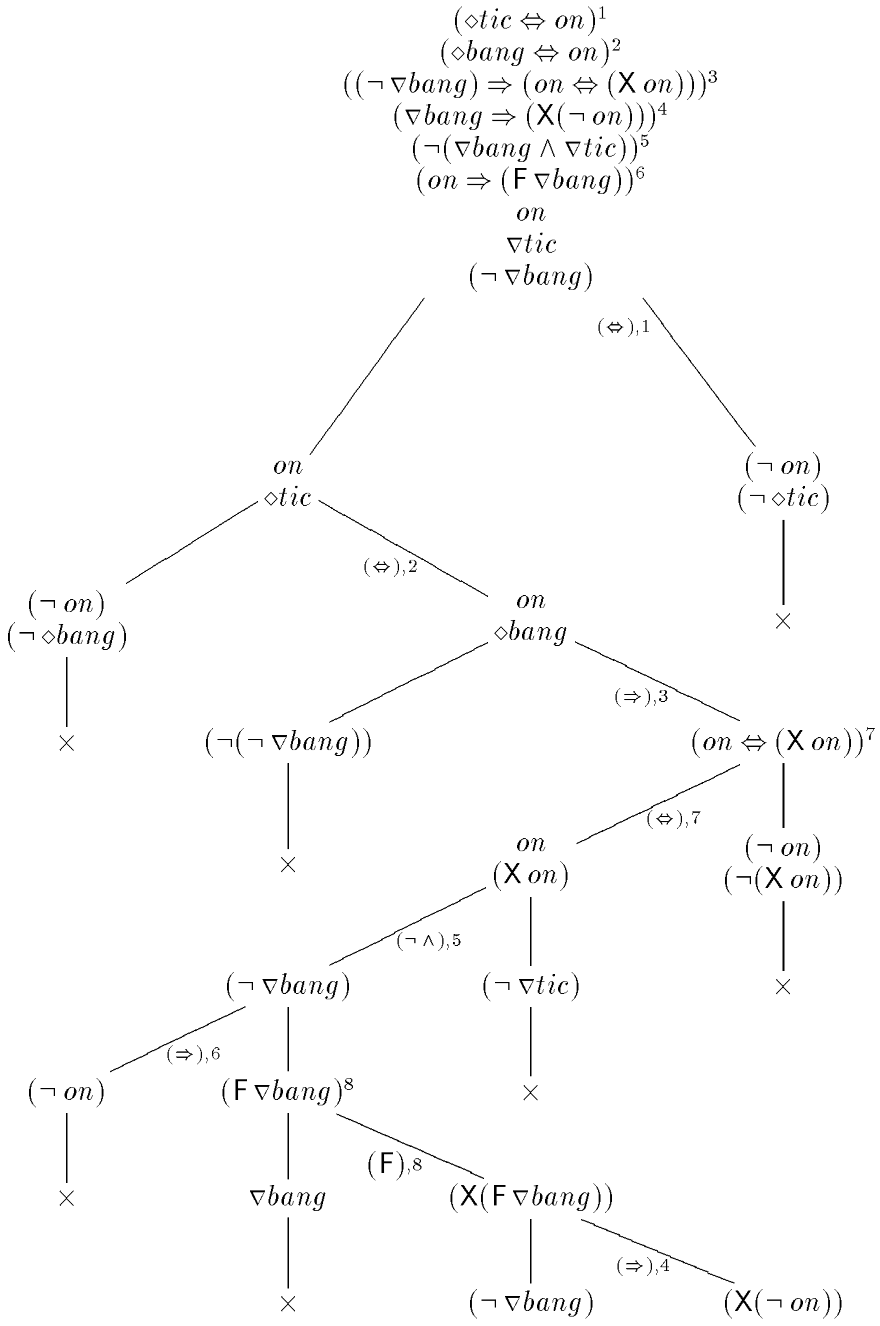
We can now take a look at how inference rules are used to build tableaux. Let us first introduce some useful notions.

We define 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*, $\theta_{11}, \ldots, \theta_{1k_1}; \ldots ; \theta_{n1}, \ldots, \theta_{nk_n} \therefore \theta$ is an inference rule and θ is in a branch of T then T', the tree which results from adding the nodes $\{\theta_{11}, \ldots, \theta_{1k_1}\}$, ..., $\{\theta_{n1}, \ldots, \theta_{nk_n}\}$ 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 to it. $\square$

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)
Consider once again the "clock_bomb" specification in example 2.8. A tableau for the set of formulae $\Gamma_{clock\_bomb} \cup \{on, \triangledown tic, (\neg \triangledown bang)\}$ is:



Note that 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.

From it we can conclude that the clock_bomb 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. This observation is supported by the next result. $\square$

Theorem 4.4 (Soundness)
A life-cycle λ over Σ satisfies at $i \in I\!N$ all the formulae in 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. $\square$

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

Theorem 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. $\square$

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 reasoning about "next state". 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 characterise 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:

- if $B \subseteq \Sigma_{obs}$, $obs(B) = B \cup \{(\neg b) : b \in (\Sigma_{obs} \setminus B)\}$;
- if $e \in 2^{\Sigma_{act}}$, $\diamond(e) = \{\diamond a : a \in e\} \cup \{(\neg \diamond a) : a \in (\Sigma_{act} \setminus e)\}$;
- if $e \in 2^{\Sigma_{act}}$, $\triangledown(e) = \{\triangledown a : a \in e\} \cup \{(\neg \triangledown a) : a \in (\Sigma_{act} \setminus e)\}$;
- if $\Phi \subseteq Form_\Sigma$, $next(\Phi) = \{\varphi : (\mathsf{X}\, \varphi) \in \Phi\}$;
- 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 over the same signature iff λ satisfies $(\mathsf{G}\,\varphi)$ (remember that the logic is non-anchored).

The synthesis method consists of 5 steps, the last of which is optional. First, given the input specification, and by a number of 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 calculated 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 intended notion of enabling. The idea is that 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 same 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 already 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 executing each of the allowed sets of prime actions and by looking at X formulae at each of its non-contradictory branches we build up new corresponding states.

The set S^1 of state candidates is inductively defined as follows:

- $\mathsf{G}(\Gamma) \cup \diamond(e) \cup obs(B) \in S^1$ for each $e \in 2^{\Sigma_{act}}$ and $B \subseteq \Sigma_{obs}$;
- $next(\Omega) \cup \diamond(e) \cup obs(B) \in S^1$ 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 $\Omega \in branches(tableau(\Theta \cup \triangledown(d)))$.

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 $\{\Lambda_1, \ldots, \Lambda_n\} = \{\Lambda \in branches(tableau(\Theta \cup \nabla(d))) : next(\Lambda) = next(\Omega)\}$. Then:

- $Ext(\delta) = \Lambda_1 \cup \ldots \cup \Lambda_n$.

The candidate set of transitions is:

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

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:

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

As was pointed out in [25] the construction of this graph consists of a search for all potential life-cycles satisfying th 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:

- $v^2(\Theta) = v^1(\Theta)$ 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 can define another approximation to the automaton, by restricting to those state candidates which establish their enablings:

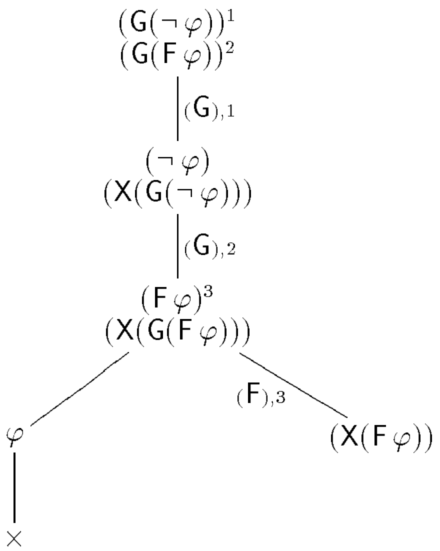
- $S^3 = S^2 \backslash T$;
- $\Delta^3 = \{\delta \in \Delta^2 : dom(\delta),\ cod(\delta) \in S^3\}$;
- $v^3(\Theta) = v^2(\Theta)$, for each $\Theta \in S^3$.

We obtain the graph $Graph^3 = (S^3, \Delta^3, dom, cod)$.

4.2.4 Step 4 - Goal construction

By now one may wonder if every life-cycle corresponding to some "computation" of $Graph^3$ satisfies all the formulae in its starting state. 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 tableau method. However, a problem arises with a particular kind of formulae, whose satisfaction at a certain instant cannot be more than eventually guaranteed. To such formulae we call eventualities. An eventuality formula is, exactly as its name suggests, a formula whose satisfaction at a certain $i \in \mathbb{N}$ 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 by life-cycle), it is easy to see that the basic constructors of eventuality formulae are those formulae of the kind $(\neg\,(\varphi \mathbin{\mathsf{U}} \psi))$. This happens because this kind of formulae require the eventual satisfaction of $(\neg\,\varphi)$. So, in general, the answer to the question above is negative. If the state we are considering includes some eventuality formula, it is possible that some computation starting there always performs transitions which delay its satisfaction. The process must then provide a way of avoiding such computations. This is the reason why we enriched the automaton with a goal structure. Eventualities occurring in any of the candidate states shall give rise to pending goals at that state and the postponing of their satisfaction is propagated through goal persistence until some transition indeed establishes it. If a state contains a formula of the referred kind, $(\neg\,\varphi)$ shall originate a goal, its fulfilment by a transition being inferred from the structure of the tableau from which it was synthesised. Due to disjunctions, we will have to consider goals as sets of formulae instead of plain formulae. Note also that, by abbreviation, the pattern $(\neg\,(\varphi \mathbin{\mathsf{U}} \psi))$ includes the usual "sometime in the future" formulae, $(\mathsf{F}\,\vartheta) \equiv (\neg((\neg\,\vartheta) \mathbin{\mathsf{U}} (\vartheta \wedge (\neg\,\vartheta))))$.

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 $Prop(\Theta) = \Theta \cup \{(\neg \nabla a) : (\neg \diamond a) \in \Theta\}$ 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 $\Lambda \in Conseq(\Theta)$. We say that a formula $(\neg \varphi)$ is a *target* of branch Λ if $(\neg \varphi) \notin \Lambda$ but there exists a formula in Λ with the pattern $(\neg(\varphi \mathsf{U} \psi))$. We denote by $targets(\Lambda)$ the set of all target formulae of Λ. □

In quite the same, 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:

- $Targets(\Theta) = \bigcup_{\Lambda \in Conseq(\Theta)} targets(\Lambda)$;
- $Relevant(\Theta) = \{\Lambda \in Conseq(\Theta) : targets(\Lambda) \neq \emptyset\}$;
- $Max(\Theta) = \{\Lambda \in Relevant(\Theta) : \{\Lambda' \in Conseq(\Theta) : targets(\Lambda) \subset targets(\Lambda')\} = \emptyset\}$.

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

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

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:

- $G^3 = \bigcup_{\Theta \in S^3} Goals(\Theta)$;

- $p^3(\Theta) = Goals(\Theta)$ 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 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:

- $g \cap Ext(\delta) \neq \emptyset$;

- $branches(tableau(Prop(cod(\delta)) \cup \{(\neg \varphi) : \varphi \in g\})) = \emptyset$.

The first condition is equivalent to saying that some formula in the goal is implied by δ, while the second condition holds if some formula in the goal is a consequence of the formulae in $cod(\delta)$. The same is to say that a transition fulfils a goal whenever some of its formulae is a consequence of the occurring action and the domain state or alternatively if some formula in the goal is a consequence of the reached 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 we obtain a G-automaton.

4.2.5 Step 5 - Goal-consistency checking (minimisation)

The problem is now reduced to checking whether or not the obtained G-automaton is goal-consistent. However, 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 synthesised from specification)
The G-automaton *synthesised* from *spec* is $syn = (\Sigma, S, \Delta, v, G, p, f)$ such that:

- $S = \{\Theta \in S^3 : p^3(\Theta) \subseteq \bigcup_{\delta \in \Delta(\Theta)} Contrib(\delta)\}$;

- $\Delta = \{\delta \in \Delta^3 : dom(\delta), cod(\delta) \in S\}$;

- $G = \bigcup_{\Theta \in S} Goals(\Theta)$;

- $p(\Theta) = p^3(\Theta)$ for each $\Theta \in S$;

- $f(\delta) = f^3(\delta)$ for each $\delta \in \Delta$. □

4.2.6 Operational semantics

Theorem 4.11 (Operational semantics)
The synthesised G-automaton syn is goal-consistent and satisfies *spec*. Moreover, for every G-automaton aut over Σ such that $aut \models spec$, we have that $Lfc(Comp_{aut}) \subseteq Lfc(Comp_{syn})$.

Proof (outline):
i) By the construction in step 5 syn is 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 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 $a \in Menu(\Theta)$. 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 eliminated in step 1 because the empty action contradicts $\triangledown a$;

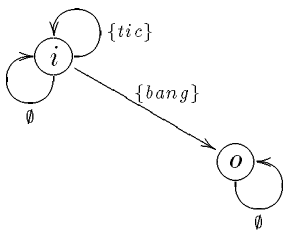
- $\varphi \equiv (\neg\,(\psi \mathsf{U} \vartheta))$
  By construction, $dom(\eta_i)$ satisfies $(\neg\,\vartheta)$ and either $(\neg\,\psi)$ or $(\mathsf{X}((\neg\,(\psi \mathsf{U} \vartheta))))$. In the first case we are done. In the second case $(\neg\,\psi)$ is a target and gives rise to goal(s).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

ii) Let $\langle s, e, r\rangle$ be a transition of aut. To prove the second statement it is enough to ensure that there is a transition $\langle s', e, r'\rangle$ of syn such that $v_{aut}(s) = v_{syn}(s')$, $v_{aut}(r) = v_{syn}(r')$, $Menu_{aut}(s) = Menu_{syn}(s')$ and $Menu_{aut}(r) = Menu_{syn}(r')$. For the purpose pick up $\Theta = \mathsf{G}(\Gamma) \cup \diamond Menu_{aut}(s) \cup obs(v_{aut}(s))$. If Θ is eliminated in step 2 then pick Θ^1 such that $\langle \Theta, e, \Theta^1\rangle$ is obtained in step 1 by adding the same observations and enablings. Proceed in this manner until either Θ^n is kept or every possibility has been rejected. Clearly the second case is not possible. It would mean that the state s of aut was incompatible with stuttering. Hence, considering an appropriate branch Ω of $tableau(\Theta^n \cup \triangledown(e))$ one gets the desired transition $\langle \Theta^n, e, next(\Omega) \cup \diamond(Menu_{aut}(r)) \cup obs(v_{aut}(r))\rangle$. □

The proof of the first statement of this theorem is a consequence of the tableaux constructions and of the essence of goal fulfilment. The second result can be obtained inductively by observing that the relevant states and transitions are not eliminated throughout the steps of the synthesis algorithm.

Let us just apply the synthesis method to a concrete example.

Example 4.12 (Operational semantics of clock_bomb system)
The G-automaton synthesised from the *clock_bomb* specification in 2.8 is represented below. Note that the goal is essential to the fulfilment of the liveness requirement expressed by axiom 5.



with:

- $i = \mathsf{G}(\Gamma_{clock\_bomb}) \cup \{(\mathsf{F}\,\nabla bang), on, \diamond tic, \diamond bang\}$,
  $o = \mathsf{G}(\Gamma_{clock\_bomb}) \cup \{(\neg\, on), (\neg \diamond tic), (\neg \diamond bang)\}$;
- $v(i) = \{on\}$, $v(o) = \emptyset$;
- $G = \{\{\nabla bang\}\}$;
- $p(i) = \{\{\nabla bang\}\}$, $p(o) = \emptyset$;
- $f(i \stackrel{\{bang\}}{\rightarrow} o) = \{\{\nabla bang\}\}$, $f(i \stackrel{\emptyset}{\rightarrow} i) = f(i \stackrel{\{tic\}}{\rightarrow} i) = f(o \stackrel{\emptyset}{\rightarrow} o) = \emptyset$. $\square$

5 Conclusion and outlook

We have shown how to synthesise 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 [31]. A first attempt in this direction can be found in [6].

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 Büchi and Muller automata on infinite objects [34]. Büchi automata are not suitable to our purposes since their acceptance condition is based on passing infinitely often in one of a number of 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 is expectedly more general (or at least 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. Either force states with no pending goals (if any) to be "frequently visited", or establish a hierarchy in the goals and ensure that those actions which contribute to their achievement occur, for instance by simply restricting the menus at each state if some goal was not yet fulfilled. The study and implementation of this strategies is a further possible extension of the present work.

A direct application of the presented transformation of temporal logic specifications into G-automata 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 [7, 15, 16, 17, 18] or in object systems [29]. In [14, 18, 28, 29] 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 open goals", "open goals which are satisfiable" and "inconsistent set of open 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 distinguished 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 set of goals. This also allows for coordination of systems behaviour based on a temporal specification without having to generate the complete system from scratch.

References

[1] J. Adámek, H. Herrlich, and G. Strecker. *Abstract and concrete categories.* John Wiley, 1990.

[2] 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.

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

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

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

[6] C. Caleiro. On the relationship between operational and denotational semantics of temporal logic specification of object behaviour. In *Proceedings of the IS-CORE Workshop 94*. World Scientific Press, 1995. To appear.

[7] 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.

[8] 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.

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

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

[11] N. Francez. *Fairness*. Springer-Verlag, 1986.

[12] 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.

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

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

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

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

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

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

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

[20] 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.

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

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

[23] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 1993. To appear.

[24] Z. Manna and A. Pnueli. A temporal proof methodology for reactive systems. In *NATO ASI Series, Series F: Computer and System Sciences*. Springer-Verlag, 1993. To appear.

[25] 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.

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

[27] N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, 1971.

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

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

[30] A. Sernadas, J. F. Costa, and C. Sernadas. An institution of object behaviour. In *Proceedings of the 9th ADT/COMPASS Workshop*, 1992.

[31] A. Sernadas and C. Sernadas. Denotational semantics of temporal object specification within an arbitrary temporal logic institution. Technical report, INESC/DMIST, 1993. Presented at IS-CORE Workshop 93 - Submitted for publication.

[32] A. Sernadas and C. Sernadas. Object certification. Technical report, INESC/DMIST, 1994. Presented at DAISD Workshop 94 - Submitted for publication.

[33] A. Sernadas, C. Sernadas, and J. F. Costa. Object specification logic. *Journal of Logic and Computation*, 1995. In print.

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