ELSEVIER

Theoretical Computer Science 267 (2001) 141–155

Theoretical Computer Science

www.elsevier.com/locate/tcs

Proving sequential function chart programs using timed automata

Dominique L'Her, Philippe Le Parc *, Lionel Marcé

Département d'informatique, Université de Bretagne Occidentale, équipe EA 2215, LIMI Langages et Interfaces pour Machines Intelligentes, 6 av. V. Le Gorgeu, BP 809, 29285 Brest cedex, France

Abstract

Applications described by sequential function chart (SFC) often being critical, we have investigated the possibilities of program checking. In particular, physical time can be handled by SFC programs using temporizations, which is why we are interested in the quantitative temporal properties. We have proposed a modeling of SFC in timed automata, a formalism which takes time into account. In this modeling, we use the physical constraints of the environment. Verification of properties can be carried out using the model-checker Kronos. We apply this method to SFC programs of average size like that of the control part of the production cell Korso. The size of the programs remains however a limit and we are studying the means of solving this problem.

Keywords: Formal methods; Checking; Timed automata; TCTL logic; Sequential function chart (SFC)

1. Introduction

The control language in which we are interested is sequential function chart [1] (SFC). Developed since 1977, this graphical language is based on the step-transition model. Through temporizations, it makes it possible to take time into account. The perfect adaptation of this intuitive and practical language to the programming of automated systems has been clearly demonstrated. It is one of the languages defined by the IEC1131-3 for the programming of Programmable Logic Controllers. For these, safety is required; it is necessary to make sure that their specifications are respected by the program. To carry out these checks, SFC has been modeled in various formalisms equipped with verification tools.

* Corresponding author.

E-mail addresses: lher@univ-brest.fr (D. L'Her), leparc@univ-brest.fr (P. Le Parc), marce@univ-brest.fr (L. Marcé).

[1] SFC is the English name of Grafcet.

0304-3975/01/$ - see front matter
PII: S0304-3975(00)00301-7

These modelings have limits: time is not taken into account. But time plays an important role in the command of many automated systems (for instance the timeouts) so it is important to treat it. This is why we are interested in temporized SFC and in its temporal verification.

After having presented the main principles of SFC, we will justify the choice of the timed automata for the modeling of SFC. Then this modeling will be described as well as the checks which it makes possible. Finally we will explain how we take into account the constraints of the physical world and how the size of the automata can be reduced.

2. SFC

SFC [2] is a chart model of the behavior of the control parts of an automated system.

2.1. Structure

The basic graphical elements are (see Fig. 1):

- The *steps* represent the various states of a system. They are symbolized by squares. The initial steps are represented by double squares. During the evolution of an SFC program, the steps are either active or inactive; during initialization, only the initial steps are active. The set of the active steps of an SFC program at a given moment defines the situation of this SFC program;
- The *transitions* are used to control moves from one state to another. They are represented by a horizontal line and control the evolution from step to step. They have two values; they can be validated or not validated. A transition is validated when all the steps preceding it are active. A receptivity is associated with each

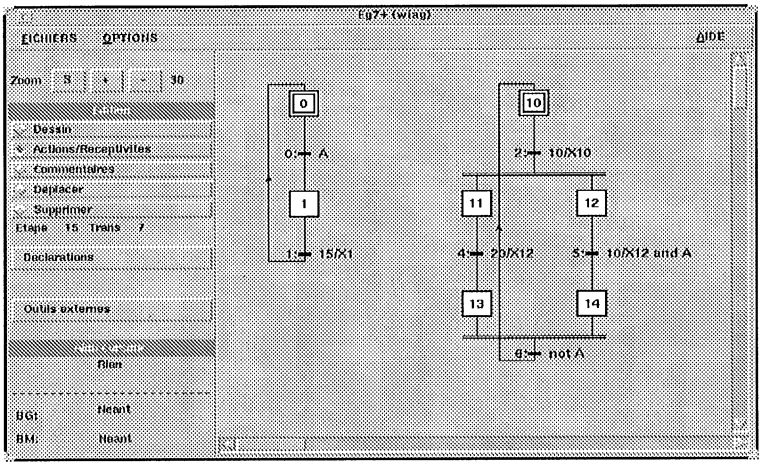

Fig. 1. An SFC program.

transition, i.e. a boolean function of the inputs and internal variables of the SFC program, for example step variables which test if a step is active or not. If a transition is validated and its receptivity has a value of true, then this transition is fireable.

Among the receptivities, a particular function makes it possible to measure time: the temporization. The temporization $t_1/X_i/t_2$ denotes a boolean condition which takes the value of true if the step i remains active at least t_1 units of time and which becomes false t_2 units of time after the deactivation of step i. No structural relation is imposed between the use of temporization and the step i referred to (in the temporization t_1/X_i, the value of t_2 is implicitly 0).

2.2. Behavior

Two postulates define the conceptual framework in which SFC must evolve:

Postulate 1: All the events are taken into account as soon as they occur and for all their incidences.

Postulate 2: In the SFC model, causality is considered with zero delay-time.

It should be noted that, as a consequence of these postulates, the SFC model is sensitive to any external event, whatever its time of occurrence. All changes in the external environment must be taken into account, and the induced reaction must be calculated with zero delay-time.

The following five rules define the evolution of an SFC program:

Rule 1: At the beginning, only the initial steps are active.

Rule 2: A transition is validated if all the preceding steps are active. A transition is fireable when it is enabled and its receptivity has the value of true.

Rule 3: A fireable transition is immediately fired. The immediately following steps are then activated and the immediately preceding steps are deactivated. Activations and deactivations are performed simultaneously.

Rule 4: If, in an SFC program, several transitions are simultaneously fireable, they are fired simultaneously.

Rule 5: If a step is simultaneously activated and deactivated, it remains active. The priority is given to activation.

2.3. Interpretation

The behavior of an SFC program is described by the five rules of evolution. Those are supplemented by interpretation algorithms. The main interpretations are named no search for stability (NSS) and search for stability (SS).

NSS interpretation: In the case of the NSS interpretation, an evolutionary step corresponds to a simple evolution, that is the simultaneous firing of all the fireable transitions. Carrying out a simple evolution step corresponds to the acquisition of inputs, to the computation of the new situation and to its output towards the external world.

SS interpretation: In the case of the SS interpretation, an evolutionary step corresponds to an iterated evolution, that is, a simple evolution with acquisition of the inputs followed by a continuation, possibly empty, of simple evolutions without acquisition

of inputs, until a stable situation is obtained. A situation is stable when no transition is fireable without new input being taken. A cycle of instability is a sequence of simple evolutions not leading to a stable situation.

Despite rules and interpretations, ambiguities still persist in the description of SFC programs. For the following modelings, the choices which were made are detailed in [11].

The SFC program of Fig. 1 will illustrate the various points of our talk. At the beginning, steps 0 and 10 are active and it is supposed that input A is false. Transitions 0 and 2 are thus validated but not fireable. Three evolutions are then possible:

- Input A becomes true before 10 units of time. In this case, transition 0 is fireable. Its firing causes the deactivation of step 0 and the activation of step 1. The situation $\{1, 10\}$ is reached, applying rule 3.
- 10 units of time run out without A becoming true. Transition 2 is fired, the situation $\{0, 11, 12\}$ is reached, applying rule 3.
- Input A becomes true exactly 10 units of time after the activation of step 10. Transitions 0 and 2 are simultaneously fired. The situation reached is $\{1, 11, 12\}$, applying rules 3 and 4.

The SFC program then continues to evolve from the current situation.

3. Checking by using timed automata

"Synchronous" languages have been proposed to answer the problems of safe programming. The basic assumption of these languages stipulates that the outputs be considered simultaneously with the inputs that generate them. The SFC language also makes this assumption. In the case of the languages Signal [8] and Lustre [6], the data flow approach still accentuates the proximity between SFC and these languages. On the other hand, the definition of SFC is purely textual and does not provide clear semantics, whereas languages such as Signal and Lustre have mathematical semantics. Therefore, the modeling of SFC in such languages [9] gives us a means of clarifying the semantic choices for SFC. This also allows us to build a simulator and to check properties. However, for checking quantitative temporal properties, this approach is not suitable. Indeed, the discrete representation of time induces an explosion of the number of states of the graph representing all possible runs, so that verifications cannot be performed in a short time.

In order to solve this problem of explosion, [5] proposes an approximative method based on convex polyhedrons. Verifications are not performed on the whole graph but on an abstraction.

We have chosen another approach which takes physical time into account. Thus we have studied timed Petri nets [4], timed transition models (TTM) [13], timed automata [7, 12] and hybrid systems [1]. Timed automata give us a good compromise between the power of expression and the possibility of verification.

3.1. Timed automata

Informally, timed automata are automata extended by a set of real variables, called clocks, the values of which grow uniformly with the passage of time and which can be set to zero. Constraints relating to these clocks are associated with the states and transitions. These timing constraints define the time during which the system can remain in a state and the possibility of firing a transition. Timed automata thus allow a compact modeling of time. Moreover, verifications using model-checking are possible on timed automata. This is why we have chosen them to model SFC.

3.1.1. Definition

A timed automaton is a quintuple $(S,\ s_{\text{init}}, H, A, Inv)$ where

- S is a finite set of control locations where s_{init} is the initial location.
- H is a finite set of clocks, real variables taking their values in the set of positive real numbers.
- A is a finite set of edges. Each edge is defined by a quintuple $(s_s,\ \psi,\ l,\ R,\ s_b)$ where s_s and s_b are the source and target locations, respectively, of the edge, ψ is a timing constraint which must be satisfied by the clocks to fire the edge, l is a label and R is the set of the clocks to be set to zero when the edge is fired. The edge $(s_s,\ \psi,\ l,\ R,\ s_b)$ is also noted $s_s \xrightarrow{\psi,l,R} s_b$.
- Inv: $S \to \psi(\mathrm{H})$ associates with each location a time-progress condition called invariant. While the clocks satisfy the invariant, the system may stay in the location.

At the beginning, the system is at the initial location with all the clocks having the value 0.

3.1.2. Semantics

The timed automaton semantics is given by $\langle Q,\ \to,\ (s_0, v_0)\rangle$ a transition system where Q is the set of states, $\to$ the set of transitions and (s_0, v_0) the initial state.

- A state (s, v) is a location s and a valuation v of all the clocks.
- The initial state is the pair (s_0, v_0) where s_0 is the initial location and v_0 is the valuation which associates 0 with all the clocks.
- From the state (s_s, v), the transition $(s_s,\ \phi,\ l, R,\ s_b)$ can be fired if the clock valuation satisfies ϕ. We note $v[R]$ the clock valuation after the firing of the transition which associates 0 to the clocks in R. The clocks in R are set to zero, the values of other clocks remaining unchanged. This behavior is expressed by the following rule:

$$\text{rule 1:}\quad \frac{s \xrightarrow{\phi,a,R} s' \wedge\ \phi(v)}{(s,v) \xrightarrow{a} (s', v[R])}$$

While the constraint associated with a state is true, the system is allowed to stay in the state. This property leads to the next rule

$$\text{rule 2:}\quad \frac{\forall t \in [v, v+d]\ Inv(s,t)}{(s,v) \xrightarrow{a} (s, v+d)}$$

At any state, the system can evolve either by a discrete state change corresponding to a move through an edge that may change the location and reset some of the clocks, or by a continuous state change due to the progress of time at a location.

3.2. Modeling

First we present the modeling [11] in a general way. Then we specify what each element of a timed automaton represents. A location represents an SFC situation, a set of values of inputs and temporizations. The transitions correspond to a change of the inputs or to an evolution of time inducing a change of the temporization values. If these modifications imply the firing of some transitions in the SFC program, the target location represents the situation after evolution. The invariants of the states and the temporal constraints express the constraints resulting from temporizations.

3.2.1. Location

In the general case, a location of a timed automaton is defined by a situation of the SFC program, a valuation of the boolean input variables and the values of the temporizations appearing in the SFC program. Several locations of the automaton can correspond to a single situation of the SFC program.

For the SFC program of Fig. 1, if we suppose that input A is false at the beginning, the initial location is $\{0, 10, \overline{A}, \overline{tempo_1}, \overline{tempo_2}, \overline{tempo_3}, \overline{tempo_4}\}$ where $tempo_1$, $tempo_2$, $tempo_3$ and $tempo_4$ denote the temporizations 10/X10, 15/X1, 20/X12 and 10/X12, respectively.

3.2.2. Clock

A clock is associated with each step appearing in a temporization. The value of a clock is the time since when the associated step has been active or inactive.

For the SFC program of Fig. 1, 3 clocks are defined: h_{10} for the step 10 appearing in $tempo_1$, h_1 for the step 1 appearing in $tempo_2$, and h_{12} for the step 12 appearing in $tempo_3$ and in $tempo_4$.

3.2.3. Invariant associated with a location

The invariant associated with a location expresses the constraint which the clocks have to satisfy, so that no temporization changes its value in the location. First of all, we look for the relevant clocks in a location, i.e. those associated with a step, being referred to by a temporization which can change values. They correspond to the clocks which satisfy one of the two conditions:

- The clock is associated with step i, step i is active and there is a false-value temporization referring to step i. This temporization may become true.
- The clock is associated with step i, step i is inactive and there is a true-value temporization referring to step i. This temporization may become false.

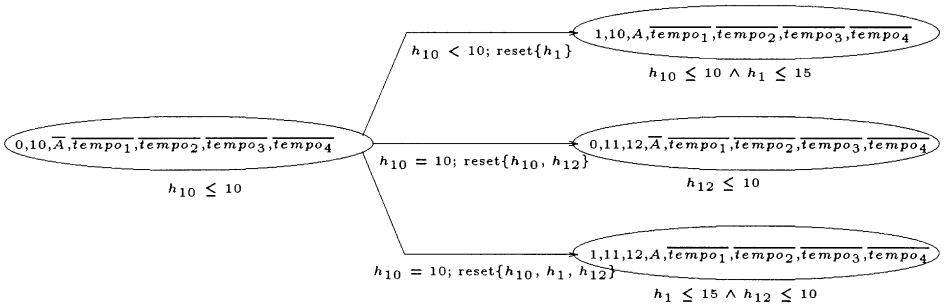

Fig. 2. First step of the construction of the timed automaton for the SFC program of Fig. 1.

The constraint associated with a clock satisfying the first condition is

$$h_i \leqslant \min_j t_{1j} \text{ for } \{tempo_j = t_{1j}/X_i/t_{2j} \text{ with } tempo_j \text{ false}\}.$$

The constraint associated with a clock satisfying the second condition is written:

$$h_i \leqslant \min_j t_{2j} \text{ for } \{tempo_j = t_{1j}/X_i/t_{2j} \text{ with } tempo_j \text{ true}\}.$$

Finally, the constraint associated with a location is true if the location does not comprise any relevant clock or, the conjunction of the constraints associated with the relevant clocks otherwise.

For instance in the initial node, only the clock h_1 is relevant because only the temporization $tempo_1$ may change. The invariant is written $h_{10} \leqslant 10$.

3.2.4. Transition

The edges of the timed automata correspond to a change of the inputs and/or an evolution of time bringing a modification of the values of temporizations. An input may change in any location. Only temporizations corresponding to the relevant clocks in the location may change.

- The *timing constraint* associated with a transition denotes whether one or more temporizations change. When the source location does not include any relevant clock, the transition is not constrained temporally: its timing constraint is "true". On the other hand if the source location includes one or more relevant clocks, then the timing constraint is a conjunction of propositions $h_i = t_i$ and $h_j < t_j$. The first form corresponds to a change of the temporization while the second denotes that the temporization remains unchanged.
- The clocks of which steps were activated (deactivated) during the transition are *set to zero* in such a way that the value of the clock is always the time since when the step has been active (inactive). For instance (Fig. 2), h_1 is set to zero on the first transition because the step 1 is activated.
- From a given situation and inputs, edges related to inputs, temporizations and edges related to steps, the new situation is obtained by a simultaneous firing of all the

fireable transitions and the new value of temporizations is computed. The *target location* is then defined by the situation reached, the inputs and temporizations being updated.

The transitions of the timed automaton do not inevitably correspond to the firing of SFC program transitions.

From the initial location $\{0, 10, \overline{A}, \overline{tempo_1}, \overline{tempo_2}, \overline{tempo_3}, \overline{tempo_4}\}$, three transitions are possible according to whether the input A and/or the temporization $tempo_1$ become true. In Fig. 2, these transitions are described.

3.2.5. Construction

The construction of the timed automaton starts with the definition of the initial location. Then this location is treated, i.e. its invariant and the transitions leaving it are computed. Then new locations are in general built. The construction continues, as long as not all the locations have been treated. Since the number of possible locations is finite the algorithm ends.

The complete automaton representing the SFC program of the Fig. 1 has 3 clocks, 14 locations and 58 transitions in the case of the SS interpretation.

3.3. Checking

We model the SFC program with a timed automaton in order to check properties using the verification tool Kronos [3].

The properties expressed on the SFC level, must be translated into timed computation tree logic (TCTL) to be checked by the model-checker Kronos. This translation is an important stage of the checking. It requires a thorough knowledge of logic and often requires very precise expression of the respective property.

3.3.1. TCTL

TCTL [7] is a temporal logic which extends arborescent logic CTL by introducing a global variable: time. As a tree logic, TCTL uses symbols that concern at the same time the set of all possible executions ($\exists$: there is an execution, $\forall$: for all the executions) and the set of execution states ($\Diamond$: there is a state, $\Box$: for all the states, U: until a state). In order to introduce time explicitly into the syntax, the scope of the temporal operators is time-limited. Thus, the formula $\forall \Box_{\leqslant 4}\, p$ intuitively means that, for all the executions of the system, proposition p is true for all the states until the fourth time unit.

3.3.2. Some properties

TCTL, although reserved to express quantitative temporal properties, makes it possible to write the usual qualitative temporal properties.

Thus to check that a situation is a deadlock, various formulae can be defined. The following formula makes it possible to know if the situation S is reachable and is always a deadlock: $(init \Rightarrow \exists \Diamond\ S) \wedge (init \Rightarrow \forall \Box\ \ (S \Rightarrow \forall \Box\ \ S))$.

Without being always a deadlock, a situation may be locked in some cases: $init \Rightarrow \exists\Diamond\ (S \wedge\ (S \Rightarrow \forall\ \Box\ S))$.

We show that the situation $\{0, 11, 14\}$ is not a deadlock but on the other hand, once the steps 11 and 14 are reached, they remain infinitely active.

On timed automata, we can also check quantitative temporal properties:

- We can check the duration of activation of a step: does step i remain active more than (at least) t units of time? For instance, we check that step 1 could remain active more than 15 units of time by showing that the following formula is true: $init \Rightarrow \exists\Diamond\ (\text{e0\_1} \wedge\ (\text{e0\_1} \Rightarrow \text{e0\_1}\ \exists U_{>15}\ \text{e0\_1}))$ where e0_1 is the proposition associated with the location when the step 1 is active.
- We can also study the time which separates two activations of distinct situations S_1 and S_2. Thus to show that between the activation of S_1 and the activation of S_2, the maximum duration is lower than t, the following formula must be false: $init \Rightarrow \exists\Diamond\ (S_1 \wedge\ (S_1 \Rightarrow (\neg S_2 \exists U_{>t} \neg S_2)))$.

Using the Kronos tool, we succeed in checking properties on the timed automata resulting from the SFC programs. By this method, we can check SFC programs of more important size and which have more temporizations. Moreover delays are not a limitation any more. Indeed, the complexity of the algorithm of verification is independent of delays.

4. Applications

We have studied the production cell Korso [10]. The programming of the control of this application is achieved very easily in an SFC program. For the checking, we have to solve two problems: taking environment into account and reducing size of the automata. We present the solutions we have found and the tools we have developed.

4.1. Taking environment into account

To explain the problem, we take an element of the operative part of Korso, the press as an example. The press consists of a horizontal plate which can move vertically. The SFC program of the press is given in Fig. 3. In steps 50 and 51, the press waits in the median position (*cap*2) until a metal blank is loaded by the robot (step 33 of the robot is then reached). Then it goes up (step 52) to the high position (*cap*3) and the metal blank is worked (step 53) during 2 units of time. Then the press goes down (step 54) to the low position (*cap*1) where it waits (step 55) to be discharged by the robot (step 41 or 42 or 43 of the robot). Finally it goes up (steps 56 and 57) until reaching its median initial position again. The process can then start again.

This SFC program is synchronized with the robot by the step variables X33, X41, X42 and X43. To study it separately, we let go these synchronizations by replacing "X33" by the variable vX33 and "X41 or X42 or X43" by the variable vX4. These variables vX33 and vX4 evolve freely.

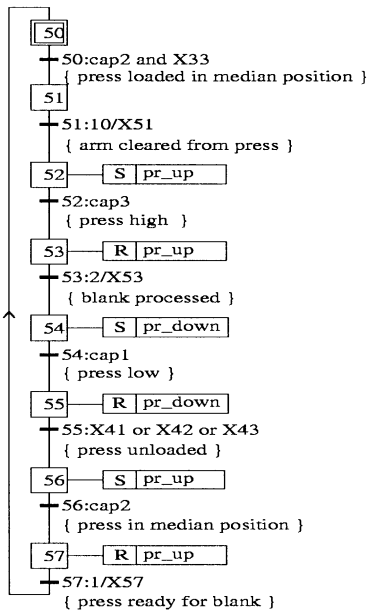

Fig. 3. SFC program of the press.

We build the corresponding timed automaton. It has 1296 locations, 262 896 transitions and 3 clocks. It is too large to be checked by the Kronos tool which accepts only automata having fewer than 65,000 transitions.

Moreover, the automaton has locations which represent the press in the high and low positions simultaneously. Under standard running, these locations have no sense; they do not fulfill the constraints of the environment. This is why the construction of the automaton was then modified so that only the locations satisfying the constraints of the environment are considered. During our study, we encountered three kinds of constraints, according to how they relate to the locations and/or the transitions:

- Only one of the sensors *cap*1, *cap*2, *cap*3 may be true at one moment because the press is in a single position. A stronger constraint can be expressed if inputs $c12$ and $c23$ (representing the position between top and medium and the position between medium and low) are introduced. In this case, it is necessary that there should be one and only one of the sensors (*cap*1, $c12$, *cap*2, $c23$, *cap*3) true at a given moment. The locations which do not satisfy this constraint are removed.
- The changes of value of the sensors are constrained to pass from the low position to the high position via a medium position. The transitions which do not satisfy this constraint are removed.
- The constraints handling, at the same time, the locations and the transitions express the links which exist between the actions and the sensors. Thus when the action *pr_up* is done, the low position is no longer reachable. In the same way when the action *pr_down* proceeds, reaching the high position is no longer possible.

We are interested in two properties of the press:

- The formula expressing that the press should not be moved to the low position if the sensor *cap*1 is true, is written: `init` $\Rightarrow \forall \Box \neg$ `(pr_down and cap1)`
- In the same way, to show that the press should not be moved to the high position if the sensor *cap*3 is true, it should be shown that the following formula is true: `init` $\Rightarrow \forall \Box \neg$ `(pr_up and cap3)`

By introducing the inputs $c12$ and $c23$ and by considering only the first two kinds of constraints, the automaton built has 922 locations and 57,606 transitions. On this automaton, the two properties are false.

Moreover, while inserting the constraints resulting from the actions, the automaton then has 314 locations and 13,670 transitions.

The first property is always false, which is due to the relaxation of synchronizations which produces an instability. Thus step 55 is not always activated; it follows that the action "*stop to go down*" is not always carried out when *cap*1 is true.

The second property is true showing that the environment has been taken into account sufficiently.

Working on a more realistic representation, we can check more properties. Taking into account the environment makes it possible to decrease the size of the automaton but does not solve all the problems of size.

4.2. Reduction of the size

During our verifications, we wish to know which situations are reachable and which values can take input in these situations. The given modeling makes it possible to answer these questions. It is however possible to consider other modelings solving this problem. If a boolean formula could be associated with a location, the most compact modeling would consist of a timed automaton reduced to the graph of the situations. As only conjunctions of the propositional variables can be associated with the locations, we cannot obtain the graph of the situations. Even so, we propose a smaller modeling than the initial modeling.

In a location, we do not denote any more the value of each input but only the value of the important inputs. For a particular situation, an input is important if a modification of its value can induce an evolution of the SFC program. In the timed automaton, a transition is defined only if it corresponds to a modification of the important inputs or a modification of temporizations.

Once the initial situation obtained, we determine the important inputs for this situation. The values of these important inputs are then fixed. The initial location is completely defined. For example, for the SFC program of Fig. 4 and NSS interpretation, the only important input for the step 0 is a. As a is initially false, the initial location is written $(0, \bar{a}, \hat{b}, \hat{c})$ where $\hat{e}$ means that the value of the input e is not of importance for the current situation.

Then, as long as there remains a state to be treated, we construct the whole automaton by the following operations: for each temporal event, for each possible combination of

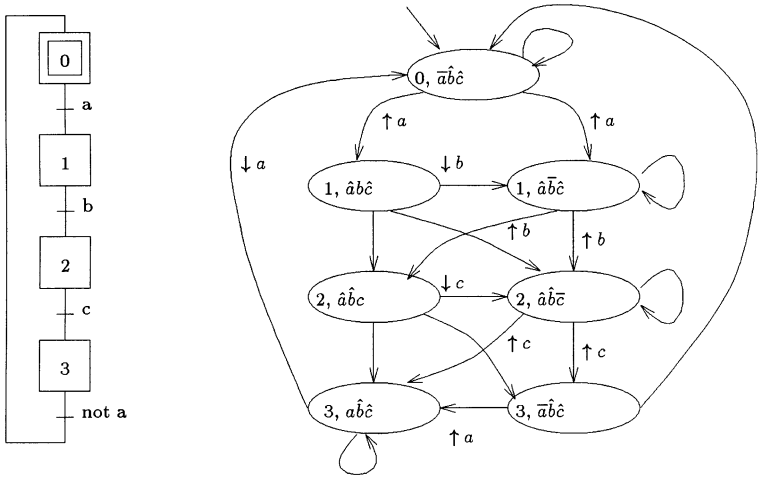

Fig. 4. SFC program and corresponding timed automaton for the new modeling in NSS interpretation (the timing constraints are not written).

the important inputs of this state, we study the target situation. If an input is important for the target location, its value must be fixed. Two cases can occur:

- It is important in the source location, its value is then perfectly defined. This is the case for the input b for the evolution from $(1, \hat{a}, b, \hat{c})$ corresponding to $\downarrow b$.
- It is not important in the source location. The target location is divided into two sets of locations, one representing the true input, the other the false input. For example, the virtual location $(2, \hat{a}, b, \hat{c})$ is reachable from the location $(1, \hat{a}, b, \hat{c})$. This location, where c is an important input, is divided into two: $(2, \hat{a}, \hat{b}, c)$ and $(2, \hat{a}, \hat{b}, \bar{c})$.

If an input is not important in the target location, then if it is important in the source location, its value is free. For example, the input b of the virtual location $(2, \hat{a}, b, \hat{c})$ is relaxed for example into $(2, \hat{a}, \hat{b}, \mathrm{c})$.

In this modeling, a location represents several locations of the preceding modeling, in the same way the number of transitions is reduced. Thus for the example and NSS interpretation, the timed automaton has 22 locations and 176 transitions in the first modeling and 7 locations and 19 transitions for the new modeling. For SS interpretation, the timed automaton has 16 locations and 112 transitions for the first modeling and 11 location and 66 transitions for the new modeling. The reduction is more sensitive for the NSS interpretation than the SS interpretation; indeed, the number of significant inputs relative to a situation is smaller in NSS than in SS.

This modeling makes it possible to considerably decrease the size of the timed automata. On the other hand, it is difficult to take environment into account with this modeling. Indeed, as it is not possible to consider boolean formulas at the level of the locations, the constrained inputs must be considered important in all situations. Therefore, in the worst case, that is to say when all the inputs are constrained, no profit will be obtained from new modeling.

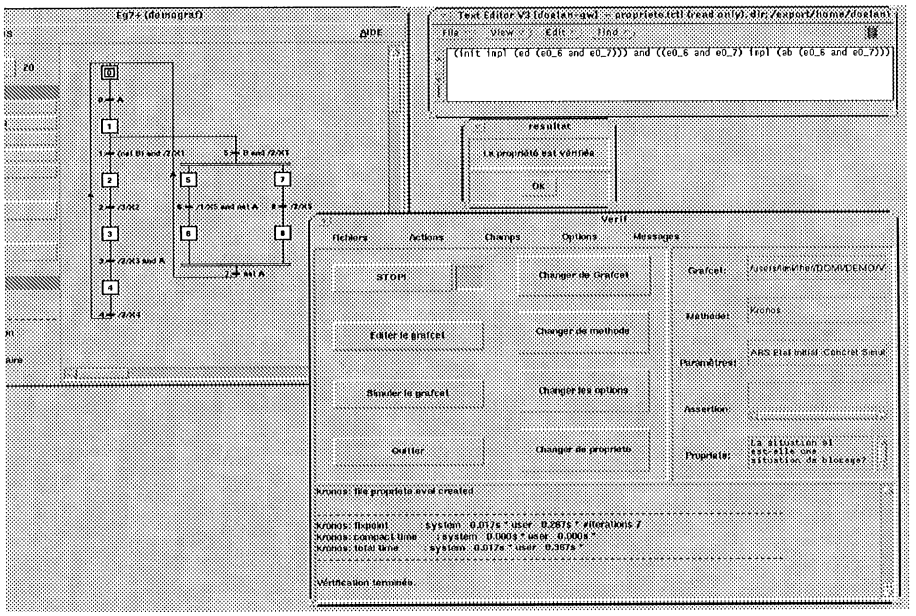

Fig. 5. Interface of verification.

We have also studied techniques permitting the decrease of the size of the systems to be checked: the composition and the abstraction. These techniques are powerful. However, for the timed systems, their study is relatively recent and few results have been obtained. Their application to the checking of SFC programs is not immediate and still requires basic work on the timed systems.

4.3. Tools

To facilitate the design and the checking, various tools have been built such as an editor of SFC progams (see Fig. 1), a simulator, the translators SFC programs-timed automata as well as an interface of verification.

The interface makes it possible to choose the parameters of the checking and to execute the chain of tools which produce the result of the checking.

It was developed in Tcl/Tk and it is composed of a control panel (see Fig. 5). From this one, the user can choose the various parameters of the verification as follows.

- The SFC program.
- The property which he wants to check. The properties are expressed in a user-friendly way. They are reachable in a tree structure.
- Options. The choice of interpretation (NSS or SS) is possible. We can moreover specify if the simultaneous modifications of inputs are authorized or not. The possibility of taking the environment into account was also given. For each type of constraint defined in the paragraph 4.1, a window of data entry has been defined.

When the checking is started, the interface takes care of several tasks:

- construction of the TCTL formula corresponding to the property,
- construction of the timed automaton corresponding to the SFC program,
- call of the tool of verification.

In Fig. 5, the result of the verification of a property on the SFC is shown.

5. Conclusion

In this work we show that it is possible to take time into account in modeling of SFC programs and to check their qualitative or quantitative temporal properties.

In modeling with the synchronous languages, we represent discrete time. During the checking, this modeling leads to a combinatorial explosion of the number of states, each instant being represented by a state.

We then have turned to timed automata. This formalism takes into account continuous time in its definition, owing to real variables called clocks. With each step i referred in a temporization $t_1/X_i/t_2$, we associate a clock. This computes the time for which the step has been active or inactive. On a timed automaton resulting from this modeling, we could check temporal properties such accessibility in a minimum or maximum time, or the durations of minimum activity and maximum. For this verification, the delays are no longer a limitation.

On the other hand, the size of the automata remains a barrier to the checking. An automaton should not have more than 65,000 transitions, so that Kronos can treat it. Unfortunately, some of the automata generated from the SFC programs can have more than 100,000 transitions. In order to solve this problem, several solutions have been investigated. Taking the environment at the level of the states and the transitions into account enables us to decrease the size of the automata considerably. In the same way, a proposed new modeling makes it possible to reduce the number of states and transitions from the automata generated. However, to increase the size of the SFC programs that can be treated, efforts must continue in these directions as in the study of the verification techniques of composition and abstraction.

To ensure more safety of SFC programs, checking only does not seem to be sufficient. In parallel, we think that a methodology should be developed making it possible to avoid design errors. This methodology could perhaps also support the building of more easily verifiable SFC programs. Furthermore it seems important to us to confront the models and theories already developed with the industrial applications.

Acknowledgements

We wish to thank the anonymous referee who helped us to improve our pidgin English.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, Theoret. Comput. Sci. 138 (1994) 3–34.
[2] N. Bouteille, P. Brard, G. Colombari, N. Cotaina, D. Richet, *Le GRAFCET*. CÉPADUÈS ÉDITION, Toulouse, France, 1992.
[3] C. Daws, A. Olivero, S. Tripakis, S. Yovine, The tool Kronos, in: R. Alur, T. Henzinger, E.D. Sontag (Eds.), DIMACS Workshop on Verification and Control of Hybrid Systems, Lecture Notes in Computer Science, vol. 1066, Springer, Berlin, 1995, pp. 208–219.
[4] C. Ghezzi, D. Mandrioli, S. Morasca, M Pezzè, A unified high-level Petri net formalism for time-critical systems, IEEE Trans. Software Eng. 17(2) (1991) 160–172.
[5] N. Halbwachs, Delay analysis in synchronous programs, in: C. Courcoubetis (Ed.), 5th Conf. Computer-Aided Verification, Lecture Notes in Computer Science, vol. 697, Springer, Berlin, 1993, pp. 333–346.
[6] N. Halbwachs, F. Lagnier, C. Ratel, Programming and verifying real-time system by means of the synchronous data-flow language Lustre, IEEE Trans. Software Eng. 18(9) (1992) 785–793.
[7] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine, Symbolic model-checking for real-time systems, Information and Computation 111(2) (1994) 193–244.
[8] P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire, Programming real time applications with Signal, Proc. IEEE 79(9) (1991) 1321–1336.
[9] P. Le Parc, D. L'Her, J.L. Scharbarg, L. Marcé, Le Grafcet revisité à l'aide d'un langage synchrone flot de données, Tech. Sci. Inform. 17(1) (1998) 63–86.
[10] C. Lewerentz, T. Lindner, Case study 'production cell': a comparative study in formal specification and verification, in: C. Lewerentz, T. Lindner (Eds.), Formal Development of Reactive Systems: Case Study Production Cell, Lecture Notes in Computer Science, vol. 891, Springer, Berlin, 1995, pp. 1–54.
[11] D. L'Her, Modélisation du Grafcet temporisé et vérification de propriétés temporelles, Ph.D. Thesis, Université de Rennes 1 (FRANCE), September 1997.
[12] X. Nicollin, J. Sifakis, S. Yovine, Compiling real-time specifications into extended automata, IEEE Trans. Software Eng. (Special Issue on Real-Time Systems) 18(9) (1992) 794–804.
[13] J.S. Ostroff, Automated verification of timed transition models, in: G. Goos, J. Hartmanis (Eds.), Workshop on Automatic Verification Methods for Finite State Systems, Lecture Notes in Computer Science, vol. 407, Springer, Berlin, 1989, pp. 247–256.