The Tool KRONOS

C. Daws * A. Olivero ** S. Tripakis * S. Yovine *

VERIMAG***
Miniparc-Zirst, Rue Lavoisier,
38330 Montbonnot St. Martin, France.

1 Introduction

KRONOS [6, 8] is a tool developed with the aim to assist the user to validate complex real-time systems. The tool checks whether a real-time system modeled by a timed automaton [4] satisfies a timing property specified by a formula of the temporal logic TCTL [3]. KRONOS implements the symbolic model-checking algorithm presented in [11], where set of states are symbolically represented by linear constraints over the clocks of the timed automaton.

In this work we present two other verification approaches we have recently implemented in KRONOS, namely forward analysis and minimization, that rely on the same symbolic representation of the state space. The emphasis is given in illustrating the interest of the two approaches rather than thoroughly presenting their technical details which can be found in [15, 7].

Forward analysis is based on the symbolic simulation of a timed automaton. By computing the set of all possible runs, starting from some given set of initial states, we can verify some interesting temporal properties on the behavior of the system, such as *reachability*, *invariance* and *bounded response*. This method turns to be in many cases more efficient than model-checking and it has the advantage that it allows error diagnosis.

Minimization consists in constructing the smallest finite quotient of the timed model with respect to a bisimulation equivalence. This method allows using timed automata not only for describing the behavior of the system but also for specifying the requirements. We can then check whether the minimal model of the system simulates or is equivalent to the one of the specification.

The paper is organized as follows. In section 2 we review timed automata. In section 3 we present the basis of symbolic forward analysis and the algorithms implemented in KRONOS, and in section 4 we apply this method for verifying the FDDI protocol [12]. In section 5 we present the minimization algorithm and in section 6 we analyze the Fischer's mutual exclusion protocol [1].

* E-mail: {Conrado.Daws,Stavros.Tripakis,Sergio.Yovine}@imag.fr. Tel: +33 76 90 96 30. Fax: +33 76 41 36 20.

** E-mail: alfredo@fing.edu.uy

*** VERIMAG is a joint laboratory of CNRS, INPG, Université Joseph Fourier, and Verilog SA, associated with IMAG.

2 Timed automata

2.1 Definition

A timed automaton is an automaton extended with a finite set of real-valued variables, called *clocks*, whose values increase uniformly with time. The timing constraints related to the system are expressed by the association of an *enabling* condition to each transition. A clock can be reset to 0 or take the value of another clock at each transition[4].

Formally, a timed automaton $\mathcal{A}$ is a tuple $\langle \mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, I \rangle$ where:

1. $\mathcal{S}$ is a finite set of *locations*.
2. $\mathcal{X}$ is a finite set of *clocks*. A *valuation* $v \in \mathcal{V}$ is a function that assigns a non-negative real-value $v(x) \in \mathsf{R}^+$ to each clock $x \in \mathcal{X}$. Let $\Psi_{\mathcal{X}}$ be the set of predicates over $\mathcal{X}$ defined as a boolean combination of atoms of the form $x \# c$ or $x - y \# c$, where $x, y \in \mathcal{X}$, $\# \in \{<, \leq, =, \geq, >\}$ and c is an integer.
3. $\mathcal{L}$ is a finite set of *labels*.
4. $\mathcal{E}$ is a finite set of *edges*. Each edge ϵ is a tuple (s, L, ψ, ρ, s') where $s \in \mathcal{S}$ is the *source*, $s' \in \mathcal{S}$ is the *target*, $L \subseteq \mathcal{L}$ are the *labels*, $\psi \in \Psi_{\mathcal{X}}$ is the *enabling* condition, and $\rho : \mathcal{X} \rightarrow \mathcal{X} \cup \{0\}$ is an assignment. We write $v[\rho]$ for the valuation v' such that for each $x \in \mathcal{X}$, if $\rho(x) = 0$ then $v'(x) = 0$, otherwise $v'(x) = v(\rho(x))$.
5. Let $\Phi_{\mathcal{X}}$ be the set of functions $\phi : \mathcal{S} \rightarrow \Psi_{\mathcal{X}}$ mapping each location s of the automaton to a predicate ψ. The invariant of $\mathcal{A}$ is a function $I \in \Phi_{\mathcal{X}}$. We write I_s for the invariant associated with s.

2.2 Semantics

A *state* of $\mathcal{A}$ is a location and a valuation of clocks satisfying the invariant associated with the location. Let $\mathcal{Q} \subseteq \mathcal{S} \times \mathcal{V}$ be the set of states of $\mathcal{A}$, that is, all pairs (s, v) such that v satisfies I_s. When $\mathcal{A}$ is in a state $(s, v) \in \mathcal{Q}$, it can evolve either by moving through an edge that changes the location and the value of some of the clocks (*discrete transition*), or by letting time pass without changing the location (*time transition*).

Discrete transitions. Let $\epsilon = (s, L, \psi, \rho, s')$. The state (s, v) has a discrete transition to (s', v'), denoted $(s, v) \xrightarrow{L}_0 (s', v')$, if $v \in \psi$ and $v' = v[\rho]$

Time transitions. Let $t \in \mathsf{R}^+$, we define $v + t$ to be the valuation v' such that $v'(x)$ is equal to $v(x) + t$ for all $x \in \mathcal{X}$. The state $(s, v) \in \mathcal{Q}$ has a time transition to $(s, v + t)$, denoted $(s, v) \xrightarrow{\emptyset}_t (s, v + t)$, if for all $t' \leq t$, $v + t' \in I_s$.

[4] The usual definition of timed automata only allows resetting clocks to 0. It has been shown in [14] that assignments of clock values does not affect decidability.

3 The forward analysis

The forward analysis verification technique is based on the computation of the symbolic runs from a given set of symbolic states. We give to the runs the structure of an oriented graph, called the *simulation graph*. Every symbolic state that appears in one or more of the runs corresponds to a single node of the graph and the simulation steps correspond to its arcs.

3.1 Symbolic runs

A *symbolic state* of the timed automaton $\mathcal{A}$ is a pair $\langle s, \psi \rangle$ where $s \in \mathcal{S}$ and $\psi \in \Psi_{\mathcal{X}}$ is a constraint such that $\emptyset \subset \psi \subseteq I_s$. The symbolic state $\langle s, \psi \rangle$ represents the set of states $(s, v) \in \mathcal{Q}$ such that $v \in \psi$. We denote $\Sigma_{\mathcal{A}}$ the set of symbolic states of $\mathcal{A}$. Let us expand the notions of discrete transition and time transition to symbolic states:

Symbolic discrete step. The discrete successor of $\langle s, \psi \rangle \in \Sigma_{\mathcal{A}}$ through the edge $\epsilon = (s, L, \psi_\epsilon, \rho, s')$ is the symbolic state $\langle s', \mathbf{post}_\epsilon(\psi) \rangle$ where $v \in \mathbf{post}_\epsilon(\psi)$ iff $\exists\, v' \in \psi \wedge \psi_\epsilon . v = v'[\rho]$. That is, the symbolic state representing the states that can be reached from some state of $\langle s, \psi \rangle$ by taking ϵ.

Symbolic time step. The time successor of $\langle s, \psi \rangle \in \Sigma_{\mathcal{A}}$ constrained by $\psi' \in \Psi_{\mathcal{X}}$ is the symbolic state $\langle s, \mathbf{post}_t^s[\psi'](\psi) \rangle$ such that $v \in \mathbf{post}_t^s[\psi'](\psi)$ iff $\exists t \geq 0, v' \in \psi$ such that $v = v' + t$ and $\forall t' . 0 \leq t' \leq t,\ v' + t' \in I_s \wedge \psi'$. That is, the symbolic state representing the states that can be reached by letting time pass from a state of $\langle s, \psi \rangle$ ensuring that ψ' continuously holds.

A *symbolic run* π of $\mathcal{A}$ starting from $\langle s, \psi \rangle \in \Sigma_{\mathcal{A}}$ and constrained by $\phi \in \Phi_{\mathcal{X}}$ is a sequence of symbolic states $\pi = \langle s_0 . \psi_0 \rangle \epsilon_1 \langle s_1 . \psi_1 \rangle \epsilon_2 \ldots \epsilon_i \langle s_i, \psi_i \rangle \ldots$ such that $\langle s_0, \psi_0 \rangle = \langle s, \mathbf{post}_t^s[\phi_s](\psi) \rangle$ and $\forall i \geq 1, \langle s_i, \psi_i \rangle = \langle s_i, \mathbf{post}_t^{s_i}[\phi_{s_i}](\mathbf{post}_{\epsilon_i}(\psi_{i-1})) \rangle$.

3.2 Simulation graph

The simulation graph corresponding to $\mathcal{A} = \langle \mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, I \rangle$ computed from $\mathcal{S}_I \subseteq \Sigma_{\mathcal{A}}$ and constrained by $\phi \in \Phi_{\mathcal{X}}$. is the graph $\mathcal{SG}_{\mathcal{A}}(\mathcal{S}_I, \phi) = \langle \mathcal{S}_I, \mathcal{S}_S, \mathcal{E}_S \rangle$ where the set of nodes $\mathcal{S}_S \subseteq \Sigma_{\mathcal{A}}$ and the set of edges $\mathcal{E}_S \subseteq \mathcal{S}_S \times \mathcal{E} \times \mathcal{S}_S$ are the smallest sets such that:

1. **init:** $\mathcal{S}_I \subseteq \mathcal{S}_S$ is the set of initial states.
2. **iter:** For every $\langle s, \psi \rangle \in \mathcal{S}_S$ and $\epsilon \in \mathcal{E}$ an edge with source in s and target in s', if $\psi' = \mathbf{post}_t^{s'}[\phi_{s'}](\mathbf{post}_\epsilon(\psi))$ is not empty, then $\langle s', \psi' \rangle \in \mathcal{S}_S$ and $\langle \langle s, \psi \rangle, \epsilon, \langle s', \psi' \rangle \rangle \in \mathcal{E}_S$.

3.3 Verification

Given a timed automaton $\mathcal{A}$ we consider three verification problems that can be solved by applying the forward analysis: the *reachability*, the *invariance* and the *bounded response* problems.

Reachability: The reachability problem consists in finding if there is a run of the system, starting from a state $q \in \mathcal{Q}$ satisfying I, such that Q can be reached in a time $t \# c$, and for which P holds continuously before Q is reached (where $\mathsf{I}, \mathsf{P}, \mathsf{Q} \in \Phi_{\mathcal{X}}$). This problem corresponds to checking the non-emptyness of the characteristic set of the TCTL formula $\mathsf{I} \wedge \mathsf{P} \exists \mathcal{U}_{\# c} \mathsf{Q}$.

Algorithmically, this is done by computing $\mathcal{SG}_{\mathcal{A}}(\mathsf{I} \wedge (\mathsf{z} = 0), \mathsf{P} \vee \mathsf{Q}')$ where $z \notin \mathcal{X}$ is an extra clock, and $\mathsf{Q}' = \mathsf{Q} \wedge (\mathsf{z} \# \mathsf{c})$. Each time a new symbolic state $\langle s, \psi \rangle$ is computed, if $\psi \cap \mathsf{Q}' \neq \emptyset$, the algorithm gives a symbolic run that validates the property.

Invariance: The invariance problem consists in finding if for all the runs starting from all states $q \in \mathcal{Q}$ satisfying I, the property Q holds for every state of the runs. This problem corresponds to checking the emptyness of the characteristic set of the TCTL formula $\neg(\mathsf{I} \Rightarrow \forall \Box \mathsf{Q})$ which is equivalent to $\mathsf{I} \wedge \mathtt{true} \exists \mathcal{U} \neg \mathsf{Q}$, that is, finding if $\neg \mathsf{Q}$ is not reachable from I.

If during the construction of $\mathcal{SG}_{\mathcal{A}}(\mathsf{I}, \mathtt{true})$ the algorithm finds a symbolic state $\langle s, \psi \rangle$ such that $\psi \cap \neg \mathsf{Q} \neq \emptyset$ then the algorithm exhibits a symbolic run that invalidates the invariance property.

Bounded response: The bounded response problem consists in finding if for every run starting from all states $q \in \mathcal{Q}$ satisfying I, there is a state of the run that satisfies Q in a time $t \leq c$ for a given $c \in \mathbb{N}$. This problem corresponds to checking the emptyness of the characteristic set of the TCTL formula $\neg(\mathsf{I} \Rightarrow \forall \Diamond_{\leq c} \mathsf{Q})$ which is equivalent to $\mathsf{I} \wedge \neg \mathsf{Q} \exists \mathcal{U}_{>c} \mathtt{true}$, that is, finding if there is no run where Q holds during more than c time units.

This is done by computing $\mathcal{SG}_{\mathcal{A}}(\mathsf{I} \wedge (\mathsf{z} = 0), \neg \mathsf{Q} \vee \mathsf{z} > \mathsf{c})$ where $z \notin \mathcal{X}$ is an extra clock. If a symbolic state $\langle s, \psi \rangle$ such that $\psi \cap (z > c) \neq \emptyset$ is found, then the property is not satisfied and a counter-example is provided.

4 The FDDI communication protocol

FDDI (Fiber Distributed Data Interface) [12] is a high performance fiber optic token ring Local Area Network. In this section we show the verification of the temporal mechanism that limits the possession time of the token by each station.

4.1 Protocol Description

We consider a network composed by N identical stations $S_1, \ldots, S_N$ and a ring, where the stations can communicate by *synchronous* messages with high priority and *asynchronous* messages with low priority.

Station. Each station S_i can be either waiting for the token ($\mathtt{Idle}_i$), in possession of the token and executing the synchronous transmission ($\mathtt{T}_i$, $\mathtt{ST}_i$) or in possession of the token and executing the asynchronous transmission ($\mathtt{T}_i$, $\mathtt{AT}_i$).

The two clocks a station uses to control the possession time of the token are called TRT_i (Token Rotation Timer) and THT_i (Token Holding Timer).

- TRT_i counts the time since the last reception of the token by the station. This clock is reset to zero each time the station S_i takes the token.
- THT_i counts the time since the last reception of the token, added to the time elapsed since the precedent one, given by the value of the clock TRT_i just before it is re-initialized.

When the station S_i receives the token (action $\mathtt{TT}_i$), the clock THT_i takes the value of the clock TRT_i, TRT_i is reset to zero, and the station S_i starts sending synchronous messages ($\mathtt{BS}_i$). The duration of the synchronous transmission ($\mathtt{ST}_i$) is given, for each station S_i, by a constant SA_i (Synchronous Allocation).

When the synchronous transmission ends (action $\mathtt{ES}_i$), the station has the possibility of starting the transmission of asynchronous messages (action $\mathtt{BA}_i$) if the current value of THT_i minus the time of synchronous transmission SA_i is greater than a global constant of the system called $TTRT$ (Target Token Rotation Timer). Before $THT_i - SA_i$ reaches the value $TTRT$, the station must release the token ($\mathtt{RT}_i$), ending the asynchronous transmission ($\mathtt{EA}_i$) if this one has began. The behavior of the station S_i is described by the timed automaton $\mathbf{Station}_i$ of the Figure 1(a).

Ring. The ring controls the transmission of the token between two consecutive stations S_i and S_{i+1}. There is a delay of td (Token Delay) time units, measured by the clock T, in this transmission. The Figure 1(b) shows the timed automaton **Ring** that models the ring for two stations.

System. The timed automaton that models the protocol is obtained as the parallel composition $\mathbf{FDDI}_N = \mathbf{Ring} \parallel \mathbf{Station}_1 \parallel \ldots \parallel \mathbf{Station}_N$ where the automata synchronize through the actions $\mathtt{TT}_i$ and $\mathtt{RT}_i$.

4.2 Properties verification

We verify here two properties of the FDDI protocol.

Bounded time for accessing the ring. The time elapsed within two consecutives receptions of the token by any station is bounded by a constant c_1. We can express this property in TCTL with the following formula:

$$(\mathtt{ST}_i \wedge T = 0) \Rightarrow \forall\Diamond_{\leq c_1} \mathtt{enable}(\mathtt{TT}_i) \tag{1}$$

where c_1 is equal to $TTRT + 2N.SA_i$, and $\mathtt{enable}(\mathtt{TT}_i)$ characterizes the symbolic states where the edge labeled $\mathtt{TT}_i$ is enabled.

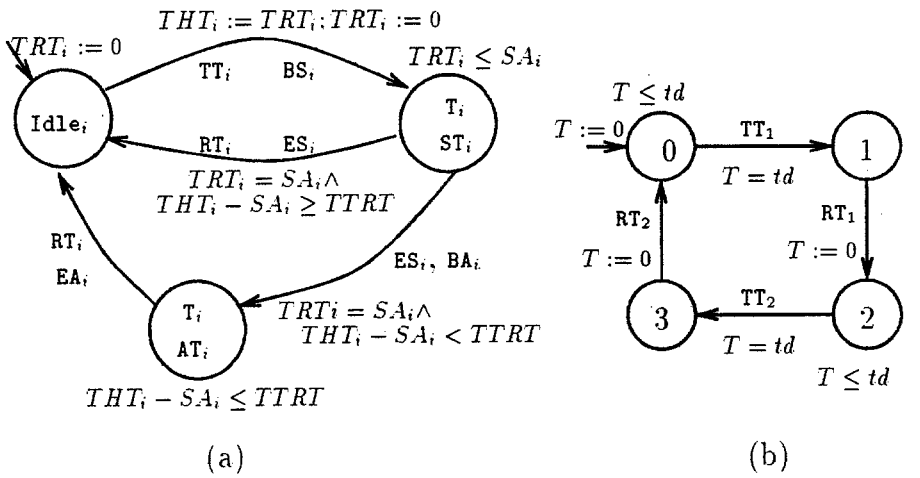

Fig. 1. Station$_i$ (a) : Ring (b)

Bounded time for sending asynchronous messages. Each idle station will send asynchronous messages before a time c_2. The formula of TCTL that describe this property is:

$$\mathtt{Idle}_i \Rightarrow \forall\Diamond_{\leq c_2}\mathtt{AT}_i \tag{2}$$

where c_2 is equal to $(N-1).TTRT + 2N.SA_i$.

Table 1 shows the results of the verification of properties (1) and (2), for different numbers of stations, applying symbolic model-checking (*backward* analysis) and symbolic simulation (*forward* analysis). We show the size of the timed automaton, the running times in seconds (*time*), the number of iterations for model-checking (*iter*) and the number of symbolic states generated for simulation (*symb*).

5 Minimization

We briefly present in this section the main ideas of the algorithm developed in [15] which is an adaptation of the minimal model generation algorithm given in [5]. In the next section we show that testing our algorithm on the Fischer's mutual exclusion protocol reveals more efficient than the minimization algorithm developed in [2].

automaton				formula	backward		forward		eval
#sta	#loc	#arcs	#clocks		time	iter	time	symb	
3	19	25	7	(1)	0.50	9	0.15	20	true
				(2)	25	29	6	1018	true
4	25	33	9	(1)	2.50	12	0.30	44	true
				(2)	3680	47	66	5522	true
5	31	41	11	(1)	10	15	1.20	92	true
				(2)	$\perp$	$\perp$	507	25532	true
6	37	49	13	(1)	61	18	3.50	188	true
7	43	57	15	(1)	435	21	10	380	true
8	49	65	17	(1)	3670	24	28	764	true
9	55	73	19	(1)	32917	27	73	1532	true
10	61	81	21	(1)	$\perp$	$\perp$	187	3068	true
11	67	89	23	(1)	$\perp$	$\perp$	483	6140	true
12	73	97	25	(1)	$\perp$	$\perp$	1123	12284	true

Table 1. Running times for different numbers of stations of the protocol FDDI with $TRTT = 50.N$, $SA_i = 20$ and $td = 0$.

5.1 Symbolic predecessors

Given a timed automaton $\mathcal{A}$, we define the notions of both discrete and time predecessors of a symbolic state as follows.

Symbolic discrete predecessor. The discrete predecessor of $\langle s', \psi' \rangle \in \Sigma_{\mathcal{A}}$ through the edge $e = (s, L, \psi_e, \rho, s')$ is the symbolic state $\langle s, \mathbf{pre}_e(\psi') \rangle$ where $v \in \mathbf{pre}_e(\psi')$ iff $v \in \psi_e$ and $v[\rho] \in \psi'$.

Symbolic time predecessor. The time predecessor of $\langle s, \psi \rangle \in \Sigma_{\mathcal{A}}$ constrained by $\psi' \in \Psi_{\mathcal{X}}$ is the symbolic state $\langle s, \mathbf{pre}_t^s[\psi'](\psi) \rangle$ such that: $v \in \mathbf{pre}_t^s[\psi'](\psi)$ iff $\exists t \geq 0.\ v + t \in \psi$ and $\forall t'.0 \leq t' \leq t.\ v + t' \in I_s \wedge \psi'$.

5.2 Partitions and bisimulations

Let Π be a partition of Q such that all classes of Π are symbolic states in $\Sigma_{\mathcal{A}}$. For $\sigma, \sigma' \in \Pi$, let $\mathbf{pre}[\sigma](\sigma')$ stand for either $\sigma \cap \mathbf{pre}_e(\sigma')$ or $\mathbf{pre}_t[\sigma](\sigma')$. We define $Succs_\Pi(\sigma) = \{\sigma' \mid \mathbf{pre}[\sigma](\sigma') \neq \emptyset\}$ and $Preds_\Pi(\sigma) = \{\sigma' \mid \mathbf{pre}[\sigma'](\sigma) \neq \emptyset\}$.

A class $\sigma \in \Pi$ is *stable* if for all $\sigma' \in \Pi$, $\mathbf{pre}[\sigma](\sigma') \in \{\sigma, \emptyset\}$, that is, either no state in σ has a discrete (resp. time) transition to a state in σ', or for all states in σ there exists a discrete (resp. time) successor in σ'. The partition Π is a *bisimulation* if every symbolic state $\sigma \in \Pi$ is stable.

5.3 Minimization algorithm

Given an initial partition Π_{init}, and a set I of initial symbolic states, our goal is to compute the coarsest bisimulation Π which is finer than Π_{init} containing only those classes which are reachable from I. Π is computed by the following algorithm.

$\Pi := \Pi_{init}$; $\Gamma := \{B \in \Pi_{init} \mid B \cap I \neq \emptyset\}$; $\Delta := \emptyset$;
while $(\exists B \in \Gamma \setminus \Delta)$ **do** {
  $C_B := Split(B, \Pi)$; (1)
  if $(C_B = \{B\})$ **then** { (2)
    $\Delta := \Delta \cup \{B\}$; $\Gamma := \Gamma \cup Succs^I_\Pi(B)$; (3)
  } **else** { (4)
    $\Gamma := \Gamma \setminus \{B\}$; $\Pi := (\Pi \setminus \{B\}) \cup C_B$; $\Delta := \Delta \setminus Preds^I_\Pi(B)$; (5)
    if $B \cap I \neq \emptyset$ **then** $\Gamma := \Gamma \cup \{C \in C_B \mid C \cap I \neq \emptyset\}$;
} }

where Γ is the set of classes accessible from I and $\Delta \subseteq \Gamma$ is the set of stable accessible classes.

The function $Split(B, \Pi)$ refines the class B with respect to the current partition (1), by choosing a class $C \in \Pi$. If B is found to be stable with respect to Π, that is, $Split(B, \Pi) = \{B\}$ (2), the all successors of B are inserted to the set of accessible classes (3), since B is accessible. If B is not stable (4), $Split(B, \Pi) = \{B_1, B_2\}$, where $B_1 = \mathbf{pre}[B](C)$, $B_2 = B \cap \overline{B_1}$, and all its predecessors become unstable (5).

6 Fischer's mutual–exclusion protocol

We describe and verify here the Fischer's mutual exclusion protocol [1]. The system is made up of n timed automata $P_1, \ldots P_n$, where P_i models the behavior of process i, along with automaton X, modeling the global variable which regulates access to the critical section (see figure 2). Δ is an upper bound on the time necessary for P_i to set X to i, after verifying that X equals 0 ; δ is a lower bound on the time that P_i has to wait before re-testing X and entering its critical section, if the value of X has not changed in the meanwhile.

Observing the behavior of the system. One expects that a correct mutual–exclusion protocol should behave as the abstract graph shown in figure 3(a). We would like to check whether this ideal model is indeed equivalent to the minimal one. Since the latter also contains irrelevant actions, such as "try_i", "setXi", as well as timed transitions, we proceed as follows : (1) we compute the minimal model of $(\|P_i)\|X$ using the adapted minimization algorithm described in the previous section ; (2) we then replace all labels different from "enter_i" and "setX0i" by

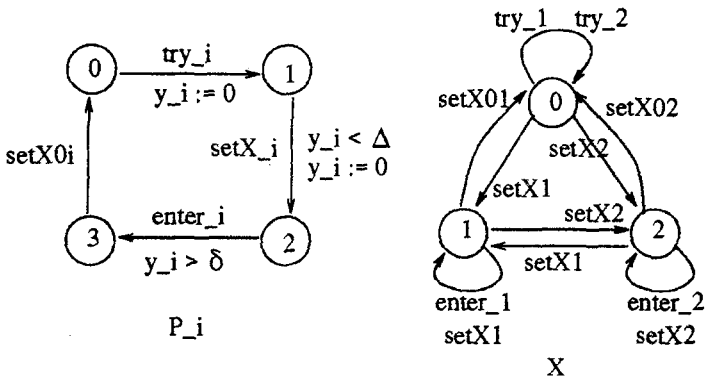

Fig. 2. Fischer's mutual–exclusion protocol specification with TA

the label τ; and (3) we further reduce the resulting graph with respect to the τ^*a bisimulation ($\approx_{\tau^*a}$), using the tool Aldebaran [9].

Performance results [5] are shown in table 2. G_{min} denotes the minimal model, and G_{τ^*a} its reduction with respect to $\approx_{\tau^*a}$. For each n, two versions of the protocol are tested : a correct one, for $\Delta = 5, \delta = 12$ (first line in the table), and an erroneous, for $\Delta = 5, \delta = 4$ (second line, marked with (*)). Figure 3(b) depicts G_{τ^*a} for $n = 2$, in the correct case [6]. It is easy to see that this graph is not equivalent to the ideal one in fig. 3(a). The reason is that the version of Fischer's protocol we have used so far permits the *starvation* of a process at control–state 2, if another process manages to get first into its critical section.

Avoiding starvation. To remedy the problem of starvation, we add an arc from state 2 to state 1, in the TA of P_i, as shown in figure 4(a). Then, we proceed as in steps (1),(2),(3) above and we find that the minimal model, for the correct case, is indeed equivalent to the ideal one in fig. 3(a). Results appear in table 3.

Comparison. The same example has been treated in [2] by minimizing a smaller TA, namely $(||P_i)||X||$Monitor, where the automaton Monitor (figure 4(b)) captures the violation of mutual exclusion, by entering an error state. Verification consists in ensuring that the minimal model contains no error state. Performance results of the two algorithms [7] are shown in table 4.

[5] We use a Sparc 10 with 128 Mbytes of main memory. $\perp$ denotes non-termination. We show only the time taken for minimization. The reduction with respect to $\approx_{\tau^*a}$ takes negligible time, except in the case marked †, where aldebaran needs 23 secs.

[6] This figure has been produced by bcg_draw [10], a tool for displaying graphs, included in the Aldebaran package.

[7] [2] have used a DEC-5100 with 40 Mbytes of main memory.

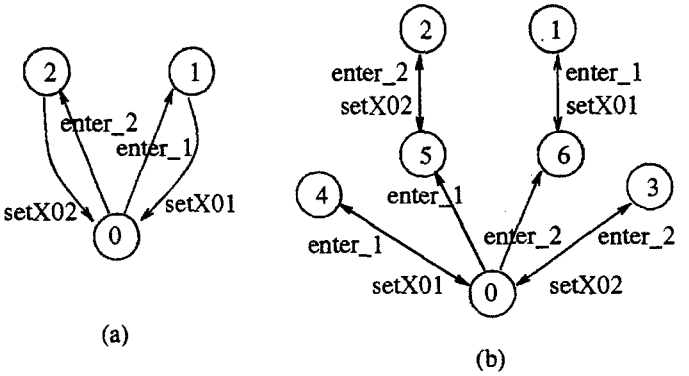

Fig. 3. Ideal mutual-exclusion (a) : Model allowing starvation (b)

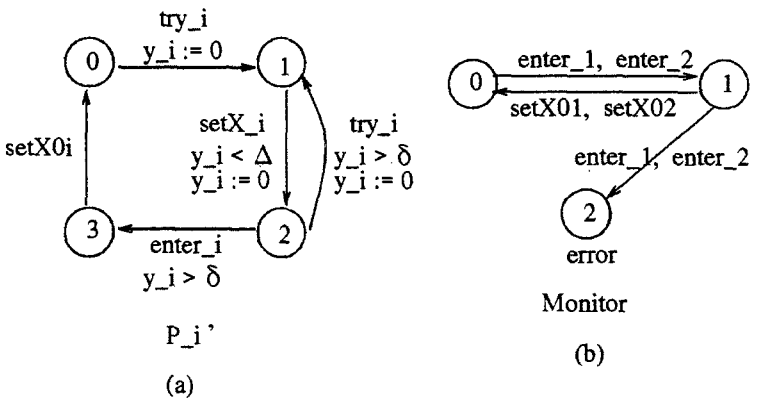

Fig. 4. Process avoiding starvation (a) ; Monitor(b)

n	composite TA		G_{min}		G_{τ^*a}		time
	states	arcs	states	trans.	states	trans.	(secs)
2	28	44	22	26	7	10	0
(*)			57	113	16	48	0
3	152	324	77	108	19	39	0.3
(*)			651	2,002	104	804	2.9
4	752	2,016	252	420	47	140	2.9
(*)			10,163	43,392	724	15,036	322 †
5	3,552	11,440	807	1,590	111	485	31.3
(*)			$\perp$	$\perp$			$\perp$

Table 2. Fischer's protocol : minimization of $(\|P_i)\|\backslash X$

n	composite TA		G_{min}		G_{τ^*a}		time
	states	arcs	states	trans.	states	trans.	(secs)
2	28	48	22	30	3	4	0
3	152	360	77	132	4	6	0
4	752	2.240	252	524	5	8	2
5	3.552	12.640	807	1.990	6	10	24

Table 3. Fischer's protocol : minimization of $(\|P_i'\)\|X$ (no starvation)

n	composite TA		G_{min}		G_{τ^*a}		time (secs)	
	states	arcs	states	trans.	states	trans.		[2]
2	24	34	22	26	7	10	0	1
(*)			47	85	12	27	0	3
3	119	213	77	108	19	39	0.2	8
(*)			402	1.117	47	205	1.5	887
4	548	1.164	252	420	47	140	2.1	192
(*)			4.437	17.902	174	1.333	40.4	$\perp$
5	2.402	5.850	807	1.590	111	485	16.3	not tested
(*)			$\perp$	$\perp$			$\perp$	not tested

Table 4. Fischer's protocol : minimization of $(\|P_i\)\|X\|$Monitor

7 Conclusion

Both approaches presented in this paper considerably improve KRONOS performance and functionalities [8].

Forward analysis permits handling examples with a large number of clocks, as the example of the FDDI protocol shows : up to 25 clocks, which, to our knowledge, exceeds the clock-space dimension of similar examples treated in the literature. Moreover, this method is capable of providing a counter-example sequence, as a diagnosis in the case a system fails to verify an invariance or bounded response property.

Minimization considerably reduces the number of states and transitions of large systems, as the example of Fischer's protocol illustrates. It also allows for further analysis, using standard techniques for untimed systems, such as comparison and reduction with respect to behavioral equivalences. The combination of timed and untimed minimization allowed us to discover the problem of starvation in the first version of the mutual-exclusion protocol.

[8] For information on how to obtain the tool, please contact the authors.

We stress the fact that Fischer's protocol has been analyzed many times, using other real-time verification tools, in particular in [2, 13]. None of these two analyses, however, deals with starvation, while the versions of the protocol used are simpler.

Finally, both forward analysis and minimization prove helpful not only for verification but also for revealing intrinsic problems of modelization, thus giving better insight to the system analyzed.

References

1. M. Abadi and L. Lamport. An old-fashioned recipe for real-time. In *Proc. REX Workshop "Real-Time: Theory in Practice"*. LNCS 600, Springer-Verlag.
2. A. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. 13th IEEE RTSS*. IEEE Computer Society Press, 1992.
3. R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.
4. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
5. A. Bouajjani, J.C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18:247–269, 1992.
6. C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In *Proc. FORTE'94*, pages 227–242, Bern, Switzerland, October 1994.
7. C. Daws and S. Yovine. Symbolic forward analysis of timed automata. Tech. Report Spectre 95-16, Verimag, Grenoble, November 1995.
8. C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. 1995 IEEE RTSS'95*, Pisa, Italy, December 1995. IEEE Computer Society Press.
9. J.Cl. Fernandez and L. Mounier. A tool set for deciding behavioural equivalences. In *CONCUR'91, Concurrency theory*. LNCS 527, Springer Verlag, August 1991.
10. H. Garavel, R. Mateescu, R. Ruffiot, and L.-P. Tock. Binary coded graphs — reference manuals of the bcg tools. Tech. Report Spectre 95-13, Verimag, Grenoble, October 1995.
11. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
12. R. Jain. *FDDI handbook: high-speed networking using fiber and other media*. Addison-Wesley, 1994.
13. K. G. Larsen, P. Petterson, and Y. Wang. Compositional and symbolic model-checking of real-time systems. In *Proc. 1995 IEEE RTSS'95*, Pisa, Italy, December 1995. IEEE Computer Society Press.
14. A. Olivero. Modélisation et analyse de systèmes temporisés et hybrides. Thèse, Institut National Polytechnique de Grenoble, Grenoble, France, September 1994.
15. S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. Tech. Report Spectre 95-15, Verimag, Grenoble, November 1995.