A Compositional Proof of a Real–Time Mutual Exclusion Protocol

Kåre J. Kristoffersen[1] *Francois Laroussinie*[3] *Kim G. Larsen*[1]
Paul Pettersson[2] *Wang Yi*[2]

[1] BRICS[†] , Aalborg University, DENMARK
[2] Department of Computer Systems, Uppsala University, SWEDEN
[3] LSV – CNRS & ENS de Cachan, FRANCE

Abstract. In this paper, we apply a compositional proof technique to an automatic verification of the correctness of Fischer's mutual exclusion protocol. It is demonstrated that the technique may avoid the state–explosion problem. Our compositional technique has recently been implemented in a tool CMC[5], which verifies the protocol for 50 processes within 172.3 seconds and using only 32MB main memory. In contrast all existing verification tools for timed systems will suffer from the state–explosion problem, and no tool has to our knowledge succeeded in verifying the protocol for more than 11 processes.

1 Introduction

It is well–known that the major problem in applying automatic verification techniques to analyze finite–state concurrent systems is the potential combinatorial explosion of the state space arising from parallel composition. In the last few years, there has been a number of automatic verification tools for real–time systems [4, 12, 8]. Experiences with these tools show that the state–explosion problem is even more serious in verifying timed systems. As such a system must satisfy certain timing constraints on its behaviour, a model–checker must keep track of not only the part of state–space explored, but also timing information associated with each state (i.e. possible clock values), which is both time and space–consuming.

During the last decade, various techniques have been developed to avoid the state–explosion problem in verifying finite–state systems, either by *symbolic* representation of the states space using BDDs [5], by application of *partial order* methods [10, 18] which suppresses unnecessary interleavings of transitions, or by application of *abstractions* and *symmetries* [6, 7, 9]. These techniques have been further extended to deal with timed systems, e.g. [4, 12],[17], [8]. However, when applying these techniques to parallel systems such as Fischer's protocol, a potential explosion in the global state–space remains. In [2], a compositional

† **B**asic **R**esearch in **C**omputer **S**cience, Centre of the Danish National Research Foundation.
[5] CMC: Compositional Model Checking

verification technique is developed by Andersen [2] for finite–state systems. In [13, 15], the technique has been further extended to deal with real–time systems modelled as networks of timed automata, which allows components of a real–time system to be gradually moved from the system description into the specification, thus avoiding any global state–space construction and even examination. Essential to the technique is that intermediate specifications are kept small using efficient minimization heuristics.

In this paper, we apply this technique to give a compositional proof for Fischer's mutual exclusion protocol. In particular, it is shown that state–explosion is avoided in the verification of the protocol: the size of the correctness proof we offer only grows polynomially in the size of the number of processes in the protocol. A similar compositional technique has recently been implemented using C++ in a tool called CMC, Compositional Model Checking. This tool gives further experimental evidence of the potential of the technique: using only 172.3 seconds and 32MB main memory CMC automatically verifies the mutual exclusion property for the acyclic version of Fischer's protocol with 50 processes.

The paper is organized as follows: In the next section we briefly introduce our modelling and specification languages for real–time systems, and the formal description of Fischer's mutual exclusion protocol. Section 3 describes the compositional quotienting method and simplification techniques for logical formulas. In section 4, we present the proof for the mutual exclusion property of Fischer's protocol. In section 5 we report on the experimental results obtained using the CMC tool and compare the performance with that of our existing tool–suite [3]. Finally, in section 6 we give some concluding remarks and illustrate future work.

2 Real–Time Systems

In this section, we briefly introduce our modelling and specification languages for real–time systems, that have been studied previously in the literature, e.g. [19, 13, 15, 16]. For details, we refer to [15].

2.1 Models

We use *timed transition systems* as a basic semantic model for real-time systems. A timed transition system is a labelled transition system with two types of labels: atomic actions and delay actions (i.e. positive reals), representing discrete and continuous changes of real-time systems. Assume a finite set of actions Act ranged over by a, b etc, and a finite set of atomic propositions $\mathcal{P}$ ranged over by p, q etc. We use $\mathbf{R}$ to stand for the set of non-negative real numbers, Δ for the set of delay actions $\{\epsilon(d) \mid d \in \mathbf{R}\}$, and L for the union $Act \cup \Delta$.

Definition 1. *A timed transition system over Act and $\mathcal{P}$ is a tuple, $\mathcal{S} = \langle S, s_0, \longrightarrow, V\rangle$, where S is a set of states, s_0 is the initial state, $\longrightarrow \subseteq S \times L \times S$ is a transition relation, and $V : S \to 2^{\mathcal{P}}$ is a proposition assignment function that for each state $s \in S$ assigns a set of atomic propositions $V(s)$ that hold in s.* □

We use synchronization functions to describe concurrency and synchronizations between timed transition systems. A *synchronization function* f is a partial function $(Act \cup \{0\}) \times (Act \cup \{0\}) \hookrightarrow Act$, where 0 denotes a distinguished no-action symbol[6]. Now, let $\mathcal{S}_i = \langle S_i, s_{i,0}, \longrightarrow_i, V_i \rangle$, $i = 1, 2$, be two timed transition systems and let f be a synchronization function. Then the *parallel composition* $\mathcal{S}_1 \,|_f\, \mathcal{S}_2$ is the timed transition system $\langle S, s_0, \longrightarrow, V \rangle$, where $s_1 \,|_f\, s_2 \in S$ whenever $s_1 \in S_1$ and $s_2 \in S_2$, $s_0 = s_{1,0} \,|_f\, s_{2,0}$, $\longrightarrow$ is inductively defined as follows:

- $s_1 \,|_f\, s_2 \xrightarrow{c} s_1' \,|_f\, s_2'$ if $s_1 \xrightarrow{a}_1 s_1'$, $s_2 \xrightarrow{b}_2 s_2'$ and $f(a,b) = c$
- $s_1 \,|_f\, s_2 \xrightarrow{\epsilon(d)} s_1' \,|_f\, s_2'$ if $s_1 \xrightarrow{\epsilon(d)}_1 s_1'$ and $s_2 \xrightarrow{\epsilon(d)}_2 s_2'$

and finally, the proposition assignment function V is defined by $V(s_1 \,|_f\, s_2) = V_1(s_1) \cup V_2(s_2)$.
The type of systems we are studying is a particular class of timed transition systems that are syntactically described by *networks of timed automata* [19, 13, 15, 16]. A timed automaton [1] is a standard finite-state automaton extended with a finite collection of real-valued clocks. Let C be a finite set of real-valued clocks ranged over by x, y etc. We use $\mathcal{B}(C)$ ranged over by g (and latter D), to stand for the set of formulas that can be an atomic constraint of the form: $x \sim n$ or $x - y \sim n$ for $x, y \in C$, $\sim \in \{\leq, \geq, <, >\}$ and n being a natural number, or a conjunction of such formulas. $\mathcal{B}(C)$ are called *clock constraints* or *clock constraint systems* over C.

Definition 2. *A timed automaton A over actions Act, atomic propositions $\mathcal{P}$ and clocks C is a tuple $\langle N, l_0, E, V \rangle$. N is a finite set of nodes (control-nodes), l_0 is the initial node, $E \subseteq N \times \mathcal{B}(C) \times Act \times 2^C \times N$ corresponds to the set of edges, and finally, $V : N \to 2^{\mathcal{P}}$ is a proposition assignment function. In the case, $\langle l, g, a, r, l' \rangle \in E$ it is written, $l \xrightarrow{g,a,r} l'$.* □

The semantics of a timed automaton is given in terms of *clock assignments*. A clock assignment u for C is a function from C to $\mathbf{R}$. Let $\mathbf{R}^C$ denote the set of clock assignments for C. For $u \in \mathbf{R}^C$, $x \in C$ and $d \in \mathbf{R}$, $u + d$ denotes the time assignment which maps each clock x in C to the value $u(x) + d$. For $C' \subseteq C$, $[C' \mapsto 0]u$ denotes the assignment for C which maps each clock in C' to the value 0 and agrees with u over $C \backslash C'$. A semantical *state* of an automaton A is a pair (l, u) where l is a node of A and u a clock assignment for C. The initial state of A is (l_0, u_0) where u_0 is the initial clock assignment mapping all clocks in C to 0. The semantics of A is given by the timed transition system $\mathcal{S}_A = \langle S, \sigma_0, \longrightarrow, V \rangle$, where S is the set of states of A, σ_0 is the initial state (l_0, u_0), $\longrightarrow$ is the transition relation defined as follows:

- $(l, u) \xrightarrow{a} (l', u')$ if there exist r, g such that $l \xrightarrow{g,a,r} l'$, $g(u)$ and $u' = [r \to 0]u$
- $(l, u) \xrightarrow{\epsilon(d)} (l', u')$ if $(l = l')$, $u' = u + d$

and V is extended to S simply by $V(l, u) = V(l)$.

---

[6] We extend the transition relation of a timed transition system such that $s \xrightarrow{0} s'$ iff $s = s'$.

$$\begin{aligned}
\langle s,u\rangle &\models c \Rightarrow c(u)\\
\langle s,u\rangle &\models p \Rightarrow p \in V(s)\\
\langle s,u\rangle &\models cp \vee \varphi \Rightarrow \langle s,u\rangle \models cp \ \text{ or } \ \langle s,u\rangle \models \varphi\\
\langle s,u\rangle &\models \varphi \wedge \psi \Rightarrow \langle s,u\rangle \models \varphi \ \text{ and } \ \langle s,u\rangle \models \psi\\
\langle s,u\rangle &\models \forall\!\!\!\forall\varphi \Rightarrow \forall d, s' :\ s \xrightarrow{\epsilon(d)} s' \ \Rightarrow\ \langle s', u+d\rangle \models \varphi\\
\langle s,u\rangle &\models [a]\,\varphi \Rightarrow \forall s' :\ s \xrightarrow{a} s' \ \Rightarrow\ \langle s', u\rangle \models \varphi\\
\langle s,u\rangle &\models x \ \mathsf{in}\ \varphi \Rightarrow \langle s,u'\rangle \models \varphi \ \text{ where } \ u' = [\{x\} \to 0]u\\
\langle s,u\rangle &\models Z \Rightarrow \langle s,u\rangle \models \mathcal{D}(Z)
\end{aligned}$$

Table 1. Definition of satisfiability.

Finally, for two timed automata A and B and a synchronization function f, the parallel composition $A\,|_f\,B$ denotes the timed transition system $\mathcal{S}_A\,|_f\,\mathcal{S}_B$.

2.2 Specifications

To specify safety and bounded liveness properties of timed systems, we use the timed modal logic $\mathcal{L}_s$, studied in [14, 15, 16]. Let K be a finite set of clocks, called formula clocks, and Id a set of identifiers. The set of formulas of $\mathcal{L}_s$ over K, Id, Act, and $\mathcal{P}$ is generated by the following syntax with φ and ψ ranging over $\mathcal{L}_s$:

$$\varphi \ ::= \ cp \ \mid \ cp \vee \varphi \ \mid \ \varphi \wedge \psi \ \mid \ \forall\!\!\!\forall\varphi \ \mid \ [a]\,\varphi \ \mid \ z \ \mathsf{in}\ \varphi \ \mid \ Z$$

where cp may be an atomic clock constraint c in the form of $x \sim n$ or $x - y \sim n$ for $x, y \in K$ and natural number n, or an atomic proposition $p \in \mathcal{P}$, $a \in Act$ (an action), $z \in K$ and $Z \in Id$ (an identifier). The meaning of the identifiers is specified by a declaration $\mathcal{D}$ assigning a formula of $\mathcal{L}_s$ to each identifier. When $\mathcal{D}$ is understood we write $Z \stackrel{\text{def}}{=} \varphi$ for $\mathcal{D}(Z) = \varphi$.

Given a timed transition system $\mathcal{S} = \langle S, s_0, \longrightarrow, V\rangle$ described by a network of timed automata, the $\mathcal{L}_s$ formulas are interpreted in terms of an extended state $\langle s, u\rangle$ where $s \in S$ is a state of a timed transition system, and u is a clock assignment for K.

Let $\mathcal{D}$ be a declaration. Formally, the satisfaction relation $\models_{\mathcal{D}}$ between extended states and formulas is defined as the largest relation satisfying the implications of Table 1. For simplicity, we shall omit the index $\mathcal{D}$ and write $\models$ instead of $\models_{\mathcal{D}}$ whenever it is understood from the context.

Finally, a network of timed automata A satisfies a formula φ written $A \models \varphi$ when $\langle (l_0, u_0), v_0\rangle \models \varphi$ where l_0 is the initial node of A, and u_0 and v_0 are the assignments with $u_0(x) = 0$ for all automaton clocks x and $v_0(z) = 0$ for all formula clocks z. Note that (l_0, u_0) is the initial state of A.

2.3 Fischer's Protocol Revisited

As an example of networks of timed automata, we study Fischer's mutual exclusion protocol. The reason for choosing this example is that it is well–known and well–studied by researchers in the context of real–time verification. More importantly, the size of the example can be easily scaled up by simply increasing

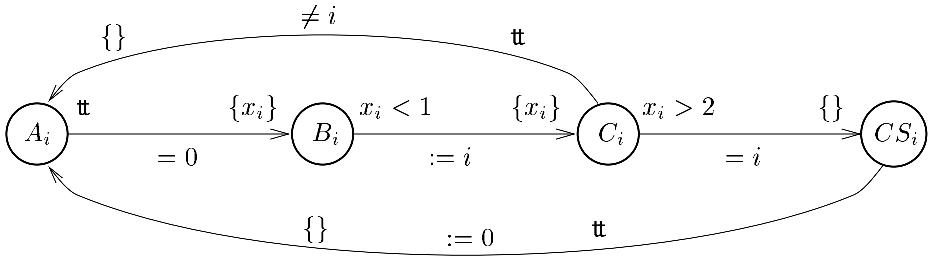

Fig. 1. Fischers Protocol for Mutual Exclusion.

the number of processes in the protocol, thus increasing the number of control–nodes — causing state–space explosion — and the number of clocks — causing region–space explosion. Thus it is particularly well–suited for our technique.

The protocol is to guarantee mutual exclusion in a concurrent system consisting of a number of processes, using clock constraints and a shared variable. We shall model each of the processes as a timed automaton, and the protocol as a network of timed automata. Each of the processes is assumed to have a local clock. The idea behind the protocol is that the timing constraints on the local clocks are set so that all processes can change the global variable to its own process number, then read the global variable later and if the shared variable is still equal to its own number, enter the critical section. Each process P_i with i being its identifier, has a clock x_i. Let $A_k = \{:= i \mid i = k+1...n\}$, $T_k = \{= i \mid i = k+1...n\}$, $F_k = \{\neq i \mid i = k+1...n\}$, and $S_k = A_k \cup T_k \cup F_k$. We model the shared variable as a timed automaton V over the set of atomic actions $S_0 \cup \{:= 0, = 0\}$, where $V = \langle N, h_0, E, V\rangle$ with $N = \{V_0...V_n\}$, $h_0 = V_0$, $E = \{\langle V_i, \mathtt{tt}, := j, \emptyset, V_j\rangle \mid i,j = 0...n\} \cup \{\langle V_i, \mathtt{tt}, = i, \emptyset, V_i\rangle \mid i = 0...n\} \cup \{\langle V_i, \mathtt{tt}, \neq j, \emptyset, V_i\rangle \mid i \neq j\}$, and we simply assume V is defined by $V(V_i) = \emptyset$ for all $i \leq n$. The automaton for a typical process P_i is shown in Fig 1.

We assume that the proposition assignment function is defined in such a way that $at(l') \in V(l)$ if $l' = l$ and $\neg at(l') \in V(l)$ if $l' \neq l$ for all nodes l and l'. Now, the whole protocol is described as the following network:

$$\mathsf{FISCHER}_n \equiv (P_1|_{f_1}(P_2|_{f_2}(P_3|_{f_3}...|_{f_{n-1}}P_n)...)|_g V$$

where $|_{f_i}$ and $|_g$ are the interleaving and full synchronization operators, induced by synchronization functions f_i and g respectively, defined by $f_i(a,0) = a$ when $a \in \{:= i, = i, \neq i\}$ and $f_i(0,a) = a$ when $a \in S_i$, and $g(a,a) = a$. Note that in $P_i|_{f_i}(P_{f_{i+1}}...)$, P_i is allowed to perform $\{:= i, = i, \neq i\}$ and the righthand side is allowed to perform all actions with indices higher than i that is, S_i.

Intuitively, the protocol behaves as follows: The constraints on the shared variable V ensure that a process must reach B–node before any process reaches C–node; otherwise, it will never move from A–node to B–node. The timing constraints on the clocks ensure that all processes in C–nodes must wait until all processes in B–nodes reach C–nodes. The last process that reaches C–node and sets V to its own identifier gets the right to enter its critical section.

We need to verify that there will never be more than one process in its critical section. An instance of this general requirement can be formalized as an invariant property: $M_{12} \equiv (\neg \mathsf{at}(\mathsf{CS}_1) \vee \neg \mathsf{at}(\mathsf{CS}_2)) \wedge \bigwedge_{a \in S_0} [a]\, M_{12} \wedge \forall\!\!\!\forall M_{12}$So we need to prove the theorem $\mathsf{FISCHER}_n \models M_{12}$

3 Compositional Model–Checking

Model–checking of real–time systems may be carried out in a symbolic fashion e.g. [11, 19]. However, when applying these techniques to parallel networks such as $\mathsf{FISCHER}_n$ a potential explosions in the global symbolic state–space may seriously hamper the technique.

In [13, 15] we presented a compositional verification technique, which allows components of a real–time system to be gradually moved from the system description into the specification, thus avoiding any global state–space construction and even examination. Essential to the technique is that intermediate specifications are kept small using efficient minimization heuristics. Our technique may be seen as a real–time extension of the compositional technique presented and experimentally applied by Andersen [2] for ordinary finite–state systems. In this section we give a brief review of the technique in [13, 15].

3.1 Quotient Construction

The main ingredient in our compositional verification technique is the so–called *quotient* construction, which allows components of a network to be moved into the specification. More precisely, given a formula φ, and two timed automata A and B we may construct a formula (called the *quotient*) $\varphi /_{\!f} B$ such that

$$A \,|_f\, B \models \varphi \quad \text{if and only if} \quad A \models \varphi /_{\!f} B \tag{1}$$

The bi–implication indicates that we are moving parts of the parallel system into the formula. Clearly, if the quotient is not much larger than the original formula, we have simplified the task of model–checking, as the (symbolic) semantics of A is significantly smaller than that of $A \,|_f\, B$. More precisely, whenever φ is a formula over K, B is a timed automaton over C and l is a node of B, we define the quotient formula $\varphi /_{\!f}\, l$ over $C \cup K$ in Table 2 on the structure of φ [7] [8]. Note that the quotient construction for identifiers introduces new identifiers of the form X_l. The new identifiers and their definitions are collected in the (quotient) declaration $\mathcal{D}_B$. We recall from [15] the following important theorem, which justifies the construction:

[7] For $g = c_1 \wedge \ldots c_n$ a clock constraint we write $g \Rightarrow \varphi$ as an abbreviation for the formula $\neg c_1 \vee \ldots \vee \neg c_n \vee \varphi$. This is an $\mathcal{L}_s$–formula as atomic constraint are closed under negation.

[8] In the rule for $[a]\varphi$, we assume that all nodes l of a timed automaton are extended with a 0–edge $l \xrightarrow{\mathsf{tt},0,\emptyset} l$.

$$
\begin{aligned}
c/_f\, l &= c & p/_f\, l &= \begin{cases} \mathtt{tt} & ; p \in V(l) \\ p & ; p \notin V(l) \end{cases} \\
(\varphi_1 \wedge \varphi_2)/_f\, l &= (\varphi_1/_f\, l) \wedge (\varphi_2/_f\, l) & (\forall\!\!\forall \varphi)/_f\, l &= \forall\!\!\forall\left(\varphi/_f\, l\right) \\
(x \;\mathsf{in}\; \varphi)/_f\, l &= x \;\mathsf{in}\; (\varphi/_f\, l) & (c \vee \varphi)/_f\, l &= (c/_f\, l) \vee (\varphi/_f\, l) \\
(p \vee \varphi)/_f\, l &= (p/_f\, l) \vee (\varphi/_f\, l) & X/_f\, l &= X_l \text{ where } X_l \stackrel{\text{def}}{=} \mathcal{D}(X)/_f\, l \\
([a]\varphi)/_f\, l &= \bigwedge_{l \xrightarrow{g,c,r} l' \;\wedge\; f(b,c)=a} \left(g \Rightarrow [b](r \;\mathsf{in}\; \varphi/_f\, l')\right)
\end{aligned}
$$

Table 2. Definition of Quotient $\varphi/_f\, l$

Theorem 3. *Let A and B be two timed automata and let l_0 be the initial node of B. Then* $A\,|_f\, B \models_{\mathcal{D}} \varphi$ *if and only if* $A \models_{\mathcal{D}_B} \left(\varphi/_f\, l_0\right)$

3.2 Minimizations

It is obvious that repeated quotienting leads to an explosion in the formula (in particular in the number of identifiers). The crucial observation made by Andersen in the (untimed) finite–state case is that simple and effective transformations of the formulas in practice may lead to significant reductions.

In presence of real–time we need, in addition to the minimization strategies of Andersen, heuristics for propagating and eliminating constraints on clocks in formulas and declarations. Below we describe the transformations considered:

Reachability: When considering an initial quotient formula $\varphi/_f\, l_0$ not all identifiers in $\mathcal{D}_B$ may be reachable. Application of an "on-the-fly" technique will insure that only the reachable part of $\mathcal{D}_B$ is generated.

Boolean Simplification Formulas may be simplified using the following simple boolean equations and their duals: $\mathtt{ff} \wedge \varphi \equiv \mathtt{ff}$, $\mathtt{tt} \wedge \varphi \equiv \varphi$, $x \;\mathsf{in}\; \mathtt{ff} \equiv \mathtt{ff}$.

Constraint Propagation: Constraints on formula clocks may be propagated using various distribution laws (see Table 3). In some cases, propagation will lead to trivial clock constraints, which may be simplified to either $\mathtt{tt}$ or $\mathtt{ff}$ and hence made applicable to Boolean Simplification. As can be seen in Table 3 certain operations are to be performed on constraints during propagation. These operations include the following:

$$
\begin{aligned}
D^{\uparrow} &= \{u + d \mid u \in D \text{ and } d \in \mathbf{R}\} & \{r\}D &= \{[r \mapsto 0]u \mid u \in D\} \\
D^{\downarrow} &= \{u \mid \exists d \in \mathbf{R} :\; u + d \in D\}
\end{aligned}
$$

It may be shown that the set of constraints $\mathcal{B}(K)$ is closed under the above operations, and that they together with inclusion– and emptyness–checking may be computed efficiently (in cubic time in the number of clocks) (see e.g. [15]).

$$\emptyset \Rightarrow \varphi \equiv \mathtt{tt} \qquad D \Rightarrow c \equiv \mathtt{tt} \;\; ; \text{ if } D \subseteq c$$
$$D \Rightarrow ([a]\varphi) \equiv [a](D \Rightarrow \varphi) \qquad D \Rightarrow (\varphi_1 \wedge \varphi_2) \equiv (D \Rightarrow \varphi_1) \wedge (D \Rightarrow \varphi_2)$$
$$D \Rightarrow (x \;\mathsf{in}\; \varphi) \equiv x \;\mathsf{in}\; (\{x\}D \Rightarrow \varphi) \qquad D \Rightarrow (p \vee \varphi) \equiv p \vee (D \Rightarrow \varphi)$$
$$D \Rightarrow (c \vee \varphi) \equiv (D \wedge \neg c) \Rightarrow \varphi \qquad D \Rightarrow (\forall\!\!\!\forall \varphi) \equiv \forall\!\!\!\forall (D^{\uparrow} \Rightarrow \varphi) \;\; ; \text{ if } D^{\downarrow} \subseteq D$$
$$D \Rightarrow X \equiv D \Rightarrow \mathcal{D}(X)$$

Table 3. Constraint Propagation

Constant Propagation: Identifiers with identifier-free definitions (i.e. constants such as $\mathtt{tt}$ or $\mathtt{ff}$) may be removed while substituting their definitions in the declaration of all other identifiers.

Trivial Equation Elimination: Equations of the form $X \stackrel{\text{def}}{=} [a]X$ are easily seen to have $X = \mathtt{tt}$ as solution and may thus be removed. More generally, let S be the largest set of identifiers such that whenever $X \in S$ and $X \stackrel{\text{def}}{=} \varphi$ then $\varphi[\mathtt{tt}/S]$ [9] can be simplified to $\mathtt{tt}$. Then all identifiers of S can be removed provided the value $\mathtt{tt}$ is propagated to all uses of identifiers from S (as under Constant Propagation). The maximal set S may be efficiently computed using standard fixed point computation algorithms.

Equivalence Reduction: If two identifiers X and Y are semantically equivalent (i.e. are satisfied by the same timed transition systems) we may collapse them into a single identifier and thus obtain reduction. However, semantical equivalence is computationally very hard [10]. To obtain a cost effective strategy we approximate semantical equivalence of identifiers as follows: Let $\mathcal{R}$ be an equivalence relation on identifiers. $\mathcal{R}$ may be extended homomorphically to formulas in the obvious manner: i.e. $(\varphi_1 \wedge \varphi_2)\mathcal{R}(\vartheta_1 \wedge \vartheta_2)$ if $\varphi_1 \mathcal{R} \vartheta_1$ and $\varphi_2 \mathcal{R} \vartheta_2$, $(x \mathsf{in} \varphi)\mathcal{R}(x \mathsf{in} \vartheta)$ and $[a]\varphi\mathcal{R}[a]\vartheta$ if $\varphi\mathcal{R}\vartheta$ and so on. Now let $\cong$ be the maximal equivalence relation on identifiers such that whenever $X \cong Y$, $X \stackrel{\text{def}}{=} \varphi$ and $Y \stackrel{\text{def}}{=} \vartheta$ then $\varphi \cong \vartheta$. Then $\cong$ provides the desired cost effective approximation: whenever $X \cong Y$ then X and Y are indeed semantically equivalent. Moreover, $\cong$ may be efficiently computed using standard fixed point computation algorithms.

4 Fischers Protocol

From section 2 we recall that the protocol $\mathsf{FISCHER}_n$ consists of n processes $P_1 \dots P_n$ competing for a critical section by setting and testing a shared variable V, and that the mutual exclusion property we verify is that P_1 and P_2 cannot be in their critical section at the same time, i.e:

$$M_{12} \equiv (\neg \mathsf{at}(\mathsf{CS}_1) \vee \neg \mathsf{at}(\mathsf{CS}_2)) \wedge \textstyle\bigwedge_{a \in S_0} [a]\, M_{12} \wedge \forall\!\!\!\forall M_{12}$$

---

[9] $\varphi[\mathtt{tt}/S]$ is the formula obtained by substituting all occurrences of identifiers from S in φ with the formula $\mathtt{tt}$.

[10] For the recursion–free, untimed part of the logic semantical equivalence is already NP–complete.

In the remainder of this section we shall apply our compositional model–checking technique to verify the protocol. Our observation is that by first quotienting away V, P_1 and P_2 the quotient hereby obtained simplifies to $\mathtt{tt}$ under our minimization heuristics. Thus no examination of the components $P_3, \ldots, P_n$ is required: regardless of their behaviour the mutual exclusion property M_{12}will be satisfied. In other words, state–space explosion is avoided as it is sufficient to explore only a fixed part of the system to prove the desired property.

4.1 Constructing the Quotient

The order by which components of a network is quotiented out may highly determine the success of our method (this resembles the importance of variable–ordering in the BDD technology). Here, we choose to first quotient out the variable V followed by the relevant processes P_1 and P_2, while of course constantly minimizing the intermediate equation systems as much as possible.

Step 1: In the first step we remove the variable V from the network and transform M_{12} by quotienting it with the locations $V_0, \ldots, V_n$. This will result in an equation system containing $n+1$ identifiers $X_0, \ldots, X_n$ where X_i denotes the quotient $M_{12}/_g V_i$.

As the synchronization function g between V and the rest of the system is defined as $g(a, a) = a$ for all possible action transitions a the quotient will have exactly same conjuncts as M_{12}. Further as V does in all of its locations satisfies neither $\neg\mathsf{at}(\mathsf{CS}_1)$nor $\neg\mathsf{at}(\mathsf{CS}_2)$we get the following family of formulae X_i, where $i = 0, \ldots, n$:

$$X_i = (\neg\mathsf{at}(\mathsf{CS}_1) \vee \neg\mathsf{at}(\mathsf{CS}_2)) \wedge [=i]\, X_i \bigwedge_j [:=j]\, X_j \wedge \bigwedge_{j \neq i} [\neq j]\, X_i \wedge \forall\!\!\!\forall X_i.$$

This new equation system (i.e. the top identifier X_0) constitutes the requirement for the remaining components $P_1, \ldots, P_n$. The identifier X_i expresses the requirement to the remaining system when the variable holds the value i. That is, $(\neg\mathsf{at}(\mathsf{CS}_1) \vee \neg\mathsf{at}(\mathsf{CS}_2))$ should still be satisfied, and as long as the variable is only tested upon or as long as time passes X_i should still hold. If the variable is set to another value j the formula defined by X_j should hold instead.

Step 2: As $(\neg\mathsf{at}(\mathsf{CS}_1) \vee \neg\mathsf{at}(\mathsf{CS}_2))$is required by all identifiers and their definitions differ slightly the equation system cannot be simplified any further. Thus we proceed to transform the equation system with respect to removal of P_1 from the network. The quotient operator used to do this will be subscripted with the synchronization function f_1. In the following we will drop the synchronization function as subscript to the quotient operator, as it is obvious which one is used.

As the equation system after step 1 contains $n+1$ equations and P_1 has four control locations the new equation system will contain $4 \cdot (n+1)$ equations. For each $j = 0, \ldots, n$ we compute X_j/l, where $l \in \{A_1, B_1, C_1, CS_1\}$. The three cases where $j = 0, 1, 2$ are treated separately, while the remaining cases are treated together. When quotienting any of the identifiers X_i with A_1, B_1 or C_1 the requirement $(\neg\mathsf{at}(\mathsf{CS}_1) \vee \neg\mathsf{at}(\mathsf{CS}_2))$ disappears because $\neg\mathsf{at}(\mathsf{CS}_1)$ is satisfied in all three locations. When quotienting any of the identifiers X_i with CS_1,

($\neg\mathsf{at}(\mathsf{CS}_1)\vee\neg\mathsf{at}(\mathsf{CS}_2)$) remains in the definition of the new identifier as neither $\neg\mathsf{at}(\mathsf{CS}_1)$ nor $\neg\mathsf{at}(\mathsf{CS}_2)$ is a satisfied by CS_1. Due to lack of space we do not display this quotient, instead we continue the quotienting with respect to P_2 and therefore calculate $M_{12}/V_0/A_1/A_2$.

Step 3: The equation system of $M_{12}/V_0/A_1/A_2$ consists of $4\cdot 4\cdot(n+1)$ equations, namely the size of the product automaton of V, P_1 and P_2. The equations can be grouped as 16 equations resulting from $X_0/P_1/P_2$, 16 equations resulting from $X_1/P_1/P_2$, 16 equations resulting from $X_2/P_1/P_2$ and finally $16\cdot(n-2)$ equations resulting from $X_j/P_1/P_2$ where $j=3,\ldots,n$. For a fixed choice of locations, l_1 and l_2 in P_1 and P_2 the set of identifiers $X_j/l_1/l_2$ for $j=3,\ldots,n$ will describe very similar properties.

The equation system is presented as a *formula graph*, and part of it appears in Figure 2. Each node represents a formula identifier, and outgoing edges represents conjuncts in the definition of an identifier. For instance, the upper most node in the graph, reflects that: $X_0/A_1/A_2 = x_1 \text{ in } (X_0/B_1/A_2)\wedge [=0]\,(X_0/A_1/A_2)\wedge\ldots$. An atomic proposition (possibly a disjunction) labelling a node means that this atomic proposition appears as a conjunct in the definition of the identifier the node represents. Hence, ($\neg\mathsf{at}(\mathsf{CS}_1)\vee\neg\mathsf{at}(\mathsf{CS}_2)$) is a conjunct in the defining equation of $X_2/CS_1/CS_2$.

In the quotient all identifiers have a conjunct which refers to the identifier itself through the $\forall\!\!\!\forall$–modality. That is, For all Y the definition is on the form $Y=\ldots\wedge\forall\!\!\!\forall Y\wedge\ldots$. In the formula graph this would appear as self loops labelled with the $\forall\!\!\!\forall$–modality in all nodes, but in order to keep the graph simple we have omitted these loops. Further the quotient is symmetrical as P_1 and P_2 are symmetrical up to names on locations and clocks, therefore we only display half of the quotient as a formula graph.

To obtain a compact representation in Figure 2 we have used the the following abbreviations. A grey node labelled $X_j/l_1/l_2$ where l_1,l_2 are locations of P_1 and P_2 abbreviates the whole family of nodes $X_3/l_1/l_2,\ldots,X_n/l_1/l_2$. Similarly, edges labelled $:=j$ or $=j$ really represents a whole family of edges namely one edge for each choice of $j=3,\ldots,n$. E.g. the $:=j$ labelled edge from $X_0/A_1/A_2$ to $X_j/A_1/A_2$ in Figure 2 represents the family $X_0/A_1/A_2 \xrightarrow{:=3} X_3/A_1/A_2,\ldots,X_0/A_1/A_2 \xrightarrow{:=n} X_n/A_1/A_2$.

The overall structure of the formula graph for the resulting quotient is shown in Figure 3. Six typical parts of the quotient can be identified, these parts are labelled 1, 2, 3, 4, 5 and 6.

Part 1 of the quotient results from keeping P_2 fixed in its initial location A_2 and letting P_1 and the variable V vary as much as they can. Not surprisingly this part of the quotient reduces to $\mathtt{tt}$. We will later argue formally why this is actually the case.

Part 2 of the quotient corresponds to the behaviour part where first P_1 assigns the variable, then P_2 assigns it, where after P_2 enters the critical section and hence P_1 fails to observe the variable having the value 1 and it returns to its initial state A_1.

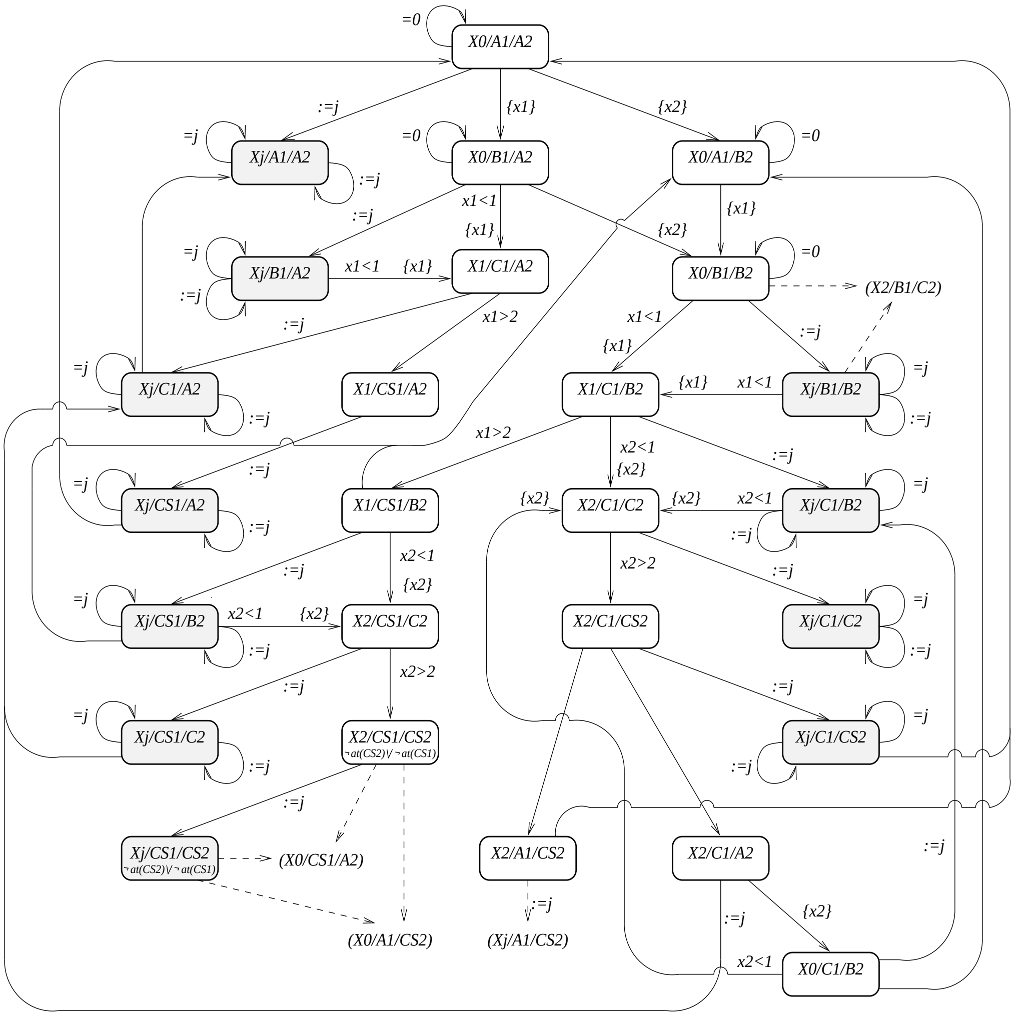

Fig. 2. Formula (sub-)Graph for $M_{12}/X_0/A_1/A_2$.

Part 3 of the quotient is where P_1 and P_2 are in the critical section at the same time. The concrete manifestation of this is that the formula identifiers of this part requires $(\neg\mathsf{at}(\mathsf{CS}_1)\vee\neg\mathsf{at}(\mathsf{CS}_2))$ to be satisfied by the remaining components $P_3,\ldots,P_n$. It is essential to the proof of the correctness that this part of the quotient will not be required to hold for the network of processes $P_3,\ldots,P_n$. The actual proof of this relies on the use of constraint propagation: We show that from the initial clock constraint (all clocks having value 0) this dangerous part of the quotient cannot be reached.

Part 4 is symmetrical to part 2 and part 5 is symmetrical to part 1. The

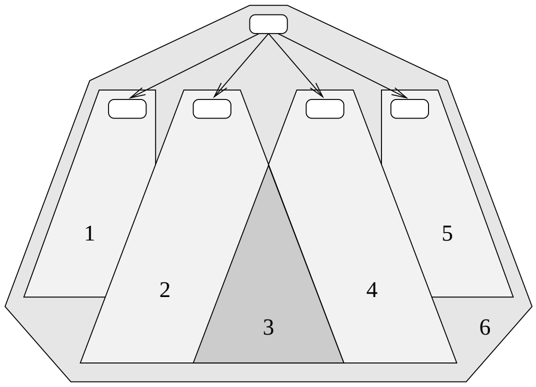

Fig. 3. Overall structure of Formula Graph

last part of the quotient, the one numbered 6, consists of the before mentioned identifiers $X_j/l_1/l_2$ where l_1 is a location in P_1, l_2 is a location in P_2 and $j = 3, \ldots, n$. This part of the quotient is the requirement when V takes a value different from $0, 1$ and 2.

4.2 Simplification

The quotient formula $M_{12}/V_0/A_1/A_2$ illustrated in figure 2 is according to Theorem 3 the necessary and sufficient property of the remaining components $P_3, \ldots, P_n$ in order that the overall system $\mathsf{FISCHER}_n$ satisfies M_{12}. We may now apply our simplification heuristics.

To our pleasant surprise we observe the quotient formula $M_{12}/V_0/A_1/A_2$ calculated above simplifies to $\mathtt{tt}$ when first applying Constraint Propagation followed by Trivial Equation Elimination. Therefore we do not have to perform quotienting with respect to the remaining components in the protocol, and hence we may conclude that an increase in the number of components in the protocol only gives rise to a *polynomial* growth in the size of the proof.

Applying constraint Propagation reveals the fact that none of the identifiers $X_j/CS_1/CS_2$ where $j = 1, \ldots, n$ can be reached from the initial constraint. As none of the remaining nodes in the graph contains propositions or clock constraints Trivial Equation Elimination will immediately reduce all identifiers and especially the top identifier $M_{12}/V_0/A_1/A_2$ to $\mathtt{tt}$.

Constraint Propagation can be implemented on our formula graphs in the following manner: Whenever $X \xrightarrow{g,\tau,r} Y$ is an edge in the graph and we consider an implication $D \Rightarrow X$, the constraint D may be propagated using the rewrite rules of Table 3 to a constraint on Y represented by the implication:

$$(\{r\}(D \wedge g))^{\uparrow} \Rightarrow Y. \tag{2}$$

Thus constraint propagation in a general formula graph, where a node can have multiple outgoing edges will result in a conjunction of formulas of the type in (2). What we intend to do here, however, is to direct the propagation of constraints along a *specific* path in the formula graph towards specific identifiers that we wish to prove unreachable. To this specific purpose we introduce the notion of *guided* Constraint Propagation. In a guided Constraint Propagation we simply focus on a specific path in the formula graph and disregard all other edges.

In the following we perform such a *guided* constraint propagation towards part 3 of the quotient by following a path through $(X_0/A_1/A_2)$, $(X_0/A_1/B_2)$, $(X_0/B_1/B_2)$, $(X_1/C_1/B_2)$, $(X_1/CS_1/B_2)$, $(X_2/CS_1/C_2)$, $(X_2/CS_1/CS_2)$, see Figure 2, and discover that $(X_2/CS_1/C_2)$ is hit by the empty constraint and thus its reference to $(X_2/CS_1/CS_2)$ has no importance in practice.

In the propagation we jump directly to the situation where the node $(X_0/B_1/B_2)$ has been reached by letting time pass while resetting first the clock x_2 and then x_1. In other words we consider the implication $(x_2 > x_1) \Rightarrow (X_0/B_1/B_2)$. Using (2) we may propagate with respect to the edge $X_0/B_1/B_2 \overset{x_1<1,\tau,\{x_1\}}{\longrightarrow} X_1/C_1/B_2$ yielding $x_2 > x_1 \Rightarrow X_1/C_1/B_2$. Now propagating this with respect to the edge $X_1/C_1/B_2 \overset{x_1>2,\tau,\emptyset}{\longrightarrow} X_2/CS_1/B_2$ yields $(x_2 > x_2 \wedge x_1 > 2) \Rightarrow X_2/CS_1/B_2$. Finally propagating this constraint with respect to $X_2/CS_1/B_2 \overset{x_2<1,\tau,\{x_2\}}{\longrightarrow} X_2/CS_1/C_2$ we get $x_2 \mathsf{~in~} (x_2 > x_1 \wedge x_1 > 2 \wedge x_2 < 1) \Rightarrow X_2/CS_1/C_2$. Clearly the constraint $(x_2 > x_1 \wedge x_1 > 2 \wedge x_2 < 1)$ is empty and hence the whole propagation simplifies to $\mathtt{tt}$.

By performing this form of guided Constraint Propagation we can prove that none of the formula identifiers in the quotient requiring P_1 or P_2 not to be in the critical section are reachable from the initial time zone. Of course we can also propagate constraints to all the other parts of the quotient, but this will not reduce the quotient as all other parts really *are* reachable.

Trivial Equation Elimination reduces all remaining identifiers to $\mathtt{tt}$ as they are defined by righthand sides which after Constraint Propagation are entirely built from the following connectives: $\mathtt{tt}, g \Rightarrow, \wedge, \forall\!\!\!\forall$.

5 Experiments

The quotient construction together with the simplification techniques presented in the previous section have been implemented with C++ in a prototype tool called CMC (Compositional Model-Checking)[11]. CMC enables us to compute the quotient of an $\mathcal{L}_s$ formula with respect to a timed automaton and then to simplify the quotient using our simplification. In fact, CMC enables quotienting with respect to formulas of the richer logic $\mathcal{L}_\nu$ [14] which allows general disjunction and existential modalities ($\exists$, $\langle a \rangle$). All simplification techniques of $\mathcal{L}_s$ can be applied (and have been implemented in CMC) to $\mathcal{L}_\nu$ with the exception that no constraint propagation has been given for general disjunction and the existential modalities.

A few new simplification strategies, which are quite useful in an actual verification, have been introduced. One of these is reduction with respect to so–called *hit–zones*, which essentially is an exhaustive constraint propagation

[11] In the near future CMC will be integrated in and available through the tool suite UPPAAL [3].

providing the automatic counterpart to the so–called guided constraint propagation used in the previous section. The idea behind this simplification is to precompute, for any variable, the domain (in terms of clock constraints) in which the variable will be considered during a given verification. Given these domains, called hit–zones, it is possible in several cases to simplify clock constraints to either 'true' or 'false' (and hence amenable to constant propagation). Another simplification which is performed by the program is to replace any variable X with the following form: $X = \ldots \wedge y < k \wedge \forall\!\!\forall X$ by 'false'.

In our experimental investigation we have compared the current version of the tool CMC with the performances of both the backward and forward reachability checker of UPPAAL on an acyclic version of Fischer's protocol. During the experiment both CMC and UPPAAL was installed on a machine running SunOS 5.5 with 32MB of primary memory and 128 of swap memory. Previously the backward reachability tool of UPPAAL has been demonstrated advantageous in a comparison with other verification tools [15] on this version of Fischer's protocol. However, as can be seen by the outcome of the present experiment in Table 4, UPPAAL is clearly outperformed by CMC, which manages verification of 50 processes.

tool \ #-processes	2	3	4	5	6	7	8	9	10	20	30	40	50
Uppaal forwards	0.2	0.2	0.9	10.7	244.4	?							
Uppaal backwards	0.1	0.2	0.3	1.2	6.2	38.5	310.6	?					
CMC	0.2	0.4	0.6	0.8	1.2	1.6	2.0	2.5	3.2	14.5	40.0	88.2	172.3

Table 4. Test results

6 Conclusion

This paper has successfully demonstrated that the compositional proof technique of [15] may avoid the state–explosion problem. In particular, it has been shown that state–explosion is avoided in the verification of Fischer's protocol: the size of the correctness proof we offer grows only polynomially in the size of the number of processes in the protocol. Furthermore, this claim has been given experimental evidence by the tool CMC, which manages verification of 50 processes. In contrast all exiting verification tools will suffer from state–explosion, and no tools has succeeded in verifying the protocol for more than 11 processes.

Immediate future work includes integration of the CMC implementation in the verification tool UPPAAL which will require certain extensions as UPPAAL allows integer variables as well as clocks with interval–bounded slopes. Also, the optimal order in which components are factored out needs a better understanding. This resembles the situation in BDDs, where the ordering of propositional variables strongly influences the size of the BDD. Our ambition for future work is to get a better understanding of when and how well our technique will work.

References

1. R. Alur and D. Dill. Automata for Modelling Real-Time Systems. *Theoretical Computer Science*, 126(2):183–236, April 1994.

2. H. R. Andersen. Partial Model Checking. *In Proc. of LICS'95*, 1995.
3. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — A Tool Suite for Symbolic and Compositional Verification of Real-Time Systems. Presented at the 1*st* Workshop on Tools and Algorithms for the Construction and Analysis of Systems, May 1995.
4. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL in 1995. In *Proc. of the 2nd Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1055 in Lecture Notes in Computer Science, pages 431–434. Springer–Verlag, March 1996.
5. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. *Logic in Computer Science*, 1990.
6. E. M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetry in Temporal Logic Model Checking. 697, 1993. In Proc. of CAV'93.
7. E. M. Clarke, O. Grümberg, and D. E. Long. Model Checking and Abstraction. *Principles of Programming Languages*, 1992.
8. C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. *In Proc. of 7th International Conference on Formal Description Techniques*, 1994.
9. E. A. Emerson and C. S. Jutla. Symmetry and Model Checking. 697, 1993. In Proc. of CAV'93.
10. P. Godefroid and P. Wolper. A Partial Approach to Model Checking. *Logic in Computer Science*, 1991.
11. Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.
12. Pei-Hsin Ho and Howard Wong-Toi. Automated Analysis of an Audio Control Protocol. In *Proc. of CAV'95*, volume 939 of *Lecture Notes in Computer Science*. Springer–Verlag, 1995.
13. F. Laroussinie and K.G. Larsen. Compositional Model Checking of Real Time Systems. In *Proc. of CONCUR '95*, Lecture Notes in Computer Science. Springer–Verlag, 1995.
14. F. Laroussinie, K.G. Larsen, and C. Weise. From Timed Automata to Logic — and Back. In *Proc. of MFCS'95*, Lecture Notes in Computer Sciencie, 1995. Also BRICS report series RS–95–2.
15. Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16th IEEE Real-Time Systems Symposium*, pages 76–87, December 1995.
16. Kim G. Larsen, Paul Pettersson, and Wang Yi. Diagnostic Model-Checking for Real-Time Systems. In *Proc. of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, Lecture Notes in Computer Science. Springer–Verlag, October 1995.
17. F. Pagani. Partial orders and verification of real-time systems. *Lecture Notes in Computer Science*, (1135), 1996.
18. A. Valmari. A Stubborn Attack on State Explosion. *Theoretical Computer Science*, 3, 1990.
19. Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. of the 7th International Conference on Formal Description Techniques*, 1994.

This article was processed using the LaTeX macro package with LLNCS style