Model–Checking for Real–Time Systems *

Kim G. Larsen[1] *Paul Pettersson*[2] *Wang Yi*[2]

[1] BRICS*** , Aalborg University, DENMARK
[2] Uppsala University, SWEDEN

Abstract. Efficient automatic model–checking algorithms for real-time systems have been obtained in recent years based on the state–region graph technique of Alur, Courcoubetis and Dill. However, these algorithms are faced with two potential types of explosion arising from parallel composition: explosion in the space of control nodes, and explosion in the region space over clock-variables.
This paper reports on work attacking these explosion problems by developing and combining *compositional* and *symbolic* model–checking techniques. The presented techniques provide the foundation for a new automatic verification tool UPPAAL . Experimental results show that UPPAAL is not only substantially faster than other real-time verification tools but also able to handle much larger systems.

1 Introduction

Within the last decade model–checking has turned out to be a useful technique for verifying temporal properties of finite–state systems. Efficient model–checking algorithms for finite–state systems have been obtained with respect to a number of logics. However, the major problem in applying model–checking even to moderate–size systems is the potential combinatorial explosion of the state space arising from parallel composition. In order to avoid this problem, algorithms have been sought that avoid exhaustive state space exploration, either by *symbolic* representation of the states space using Binary Decision Diagrams [5], by application of *partial order* methods [11, 21] which suppresses unnecessary interleavings of transitions, or by application of *abstractions* and *symmetries* [7, 8, 10].

In the last few years, model–checking has been extended to real–time systems, with time considered to be a dense linear order. A timed extension of finite automata through addition of a finite set of real–valued clock–variables has been put forward [3] (so called timed automata), and the corresponding model–checking problem has been proven decidable for a number of timed logics including timed extensions of CTL (TCTL) [2] and timed μ–calculus (T_μ) [14].

* This work has been supported by the European Communieties under CONCUR2, BRA 7166, NUTEK (Swedish Board for Technical Development) and TFR (Swedish Technical Research Council)

*** **B**asic **R**esearch in **C**omputer **S**cience, Centre of the Danish National Research Foundation.

A state of a timed automaton is of the form (l, u), where l is a control–node and u is a clock–assignment holding the current values of the clock–variables. The crucial observation made by Alur, Courcoubetis and Dill and the foundation for decidability of model–checking is that the (infinite) set of clock–assignments may effectively be partitioned into finitely many *regions* in such a way that clock–assignments within the same region induce states satisfying the same logical properties.

Model–checking of real–time systems based on the region technique suffers two potential types of explosion arising from parallel composition: *Explosion in the region space*, and *Explosion in the space of control–nodes*. We report on attacks on these problems by development and combination of two new verification techniques:

1. A *symbolic* technique reducing the verification problem to that of solving simple constraint systems (on clock–variables), and
2. A *compositional* quotient construction, which allows components of a real–time system to be gradually moved from the system into the specification. The intermediate specifications are kept small using minimization heuristics.

The property-independent nature of regions leads to an extremely fine (and large) partitioning of the set of clock–assignments. Our symbolic technique allows the partitioning to take account of the particular property to be verified and will thus in practice be considerably coarser (and smaller).

For the explosion on control–nodes, recent work by Andersen [4] on (untimed) finite–state systems gives experimental evidence that the quotient technique improves results obtained using Binary Decision Diagrams [5]. The aim of the work reported is to make this new successful compositional model–checking technique applicable to real–time systems. For example, consider the following typical model–checking problem

$$\Big(A_1 \mid \ldots \mid A_n\Big) \models \varphi$$

where the A_i's are timed automata. We want to verify that the parallel composition of these satisfies the formula φ without having to construct the complete control–node space of $(A_1 \mid \ldots \mid A_n)$. We will avoid this complete construction by removing the components A_i one by one while simultaneously transforming the formula accordingly. Thus, when removing the component A_n we will transform the formula φ into the *quotient* formula $\varphi \,/\, A_n$ such that

$$\Big(A_1 \mid \ldots \mid A_n\Big) \models \varphi \quad \text{if and only if} \quad \Big(A_1 \mid \ldots \mid A_{n-1}\Big) \models \varphi \,/\, A_n \qquad (1)$$

Now clearly, if the quotient is not much larger than the original formula we have succeeded in simplifying the problem. Repeated application of quotienting yields

$$\Big(A_1 \mid \ldots \mid A_n\Big) \models \varphi \quad \text{if and only if} \quad \mathbf{1} \models \varphi \,/\, A_n \,/ A_{n-1} \,/\, \ldots \,/\, A_1 \qquad (2)$$

where $\mathbf{1}$ is the unit with respect to parallel composition. However, these ideas alone are clearly not enough as the explosion may now occur in the size of the

final formula instead. The crucial and experimentally "verified" observation by Andersen was that each quotienting should be followed by a minimization of the formula based on a small collection of efficiently implementable strategies. In our setting, Andersen's collection is extended to include strategies for propagating and simplifying timing constraints.

We report on a new symbolic and compositional verification technique developed for the real–time logics $\mathcal{L}_\nu$ [17] and a fragment $\mathcal{L}_s$ designed specifically for expressing safety and bounded liveness properties. Comparatively less expressive than TCTL and T_μ, the fragment $\mathcal{L}_s$ is still sufficiently expressive for practical purposes allowing a number of operators of other logics to be derived. Most importantly, the somewhat restrictive expressive power of $\mathcal{L}_s$ allows for extremely efficient model–checking as demonstrated by our experimental results, which includes a comparison with other existing automatic verification tools for real–time systems (HyTech, Kronos and Epsilon).

For the logics TCTL and T_μ, [14] offers a symbolic verification technique. However, due to the high expressive power of these logics the partitioning employed in [14] is significantly finer (and larger) and implementation–wise more complicated than ours. An initial effort in applying the compositional quotienting technique to real–time systems has been given in [18].

The outline of this paper is as follows: In the next section we give a short presentation of the notions of timed automata and network. In section 3, the logic $\mathcal{L}_\nu$ and its fragment $\mathcal{L}_s$ are presented and their expressive power illustrated. Section 4 reviews region–based model–checking for $\mathcal{L}_\nu$, whereas Section 5 reports on a symbolic verification technique for the fragment $\mathcal{L}_s$ based on constraint solving. Section 6 describes the compositional quotienting technique. Finally, in Section 7 we report on our experimental results, which shows that UPPAAL is not only substantially faster than other real–time verification tools but also able to handle much larger systems.

2 Real–Time Systems

We shall use *timed transition systems* as a basic semantical model for real–time systems. The type of systems we are studying will be a particular class of timed transition systems that are syntactically described by *networks of timed automata* [22, 18].

2.1 Timed Transition 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.

Let Act be a finite set of actions ranged over by a, b etc, and $\mathcal{P}$ be a set of atomic propositions 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 actions Act and atomic propositions $\mathcal{P}$ 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.* □

Note that the above definition is standard for labelled transition systems except that we introduced a proposition assignment function V, which for each state $s \in S$ assigns a set of atomic propositions $V(s)$ that hold in s.

In order to study compositionality problems we introduce a parallel composition between timed transition systems. Following [16] we suggest a composition parameterized with a synchronization function generalizing a large range of existing notions of parallel compositions. 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 [4]. 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* $\mathcal{S}_1 \mid_f \mathcal{S}_2$ is the timed transition system $\langle S, s_0, \longrightarrow, V \rangle$, where $s_1 \mid_f s_2 \in S$ whenever $s_1 \in S_1$ and $s_2 \in S_2$, $s_0 = s_{1,0} \mid_f s_{2,0}$, $\longrightarrow$ is inductively defined as follows:

- $s_1 \mid_f s_2 \xrightarrow{c} s_1' \mid_f s_2'$ if $s_1 \xrightarrow{a}_1 s_1'$, $s_2 \xrightarrow{b}_2 s_2'$ and $f(a, b) = c$
- $s_1 \mid_f s_2 \xrightarrow{\epsilon(d)} s_1' \mid_f s_2'$ 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 $V(s_1 \mid_f s_2) = V_1(s_1) \cup V_2(s_2)$.

Note also that the set of states and the transition relation of a timed transition system may be infinite. We shall use networks of timed automata as a finite syntactical representation to describe timed transition systems.

2.2 Networks of Timed Automata

A timed automaton [3] is a standard finite–state automaton extended with a finite collection of real–valued clocks [5]. Conceptually, the clocks may be considered as the system clocks of a concurrent system. They are assumed to proceed at the same rate and measure the amount of time that has been elapsed since they were reset. The clocks values may be tested (compared with natural numbers) and reset (assigned to 0).

Definition 2. *(Clock Constraints) Let C be a set of real–valued clocks ranged over by x, y etc. We use $\mathcal{B}(C)$ to stand for the set of formulas ranged over by g, generated by the following syntax: $g ::= c \mid g \wedge g$, where c is 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. We shall call $\mathcal{B}(C)$ clock constraints or clock constraint systems over C. Moreover, $\mathcal{B}_M(C)$ denotes the subset of $\mathcal{B}(C)$ with no constant greater than M.* □

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

[5] Timed transition systems may alternatively be described using timed process calculi.

We shall use $\mathtt{tt}$ to stand for a constraint like $x \geq 0$ which is always true, and $\mathtt{ff}$ for a constraint $x < 0$ which is always false as clocks can only have non–negative values.

Definition 3. *A timed automaton A over actions Act, atomic propositions $\mathcal{P}$ and clocks C is a tuple $\langle N, l_0, E, I, V\rangle$. N is a finite set of nodes (control–nodes), l_0 is the initial node, and $E \subseteq N \times \mathcal{B}(C) \times Act \times 2^C \times N$ corresponds to the set of edges. In the case, $\langle l, g, a, r, l'\rangle \in E$ we shall write, $l \xrightarrow{g,a,r} l'$ which represents an edge from the node l to the node l' with clock constraint g (also called the enabling condition of the edge), action a to be performed and the set of clocks r to be reset. $I : N \to \mathcal{B}(C)$ is a function, which for each node assigns a clock constraint (also called the invariant condition of the node), and finally, $V : N \to 2^{\mathcal{P}}$ is a proposition assignment function which for each node gives a set of atomic propositions true in the node.* □

Note that for each node l, there is an invariant condition $I(l)$ which is a clock constraint. Intuitively, this constraint must be satisfied by the system clocks whenever the system is operating in that particular control–node.

Informally, the system starts at node l_0 with all its clocks initialized to 0. The values of the clocks increase synchronously with time at node l as long as they satisfy the invariant condition $I(l)$. At any time, the automaton can change node by following an edge $l \xrightarrow{g,a,r} l'$ provided the current values of the clocks satisfy the enabling condition g. With this transition the clocks in r get reset to 0.

Example 1. Consider the automata A_m, B_n and $C_{m,n}$ in Figure 1 where m, n, m' and n' are natural numbers used as parameters. The automaton $C_{m,n}$ has four nodes, l_0, l_1, l_2 and l_3, two clocks x and y, and three edges. The edge between l_1 and l_2 has b as action, $\{x, y\}$ as reset set and the enabling condition for the edge is $x > m$. The invariant conditions for nodes l_1 and l_2 are $x \leq m'$ and $y \leq n'$ respectively. □

Now we introduce the notion of a *clock assignment*. Formally, a clock assignment u for C is a function from C to $\mathbf{R}$. We denote by $\mathbf{R}^C$ 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 $C \backslash C'$. Whenever $u \in \mathbf{R}^C$, $v \in \mathbf{R}^K$ and C and K are disjoint, we use uv to denote the clock assignment over $C \cup K$ such that $(uv)(x) = u(x)$ if $x \in C$ and $(uv)(x) = v(x)$ if $x \in K$. Given a clock constraint $g \in \mathcal{B}(C)$ and a clock assignment $u \in \mathbf{R}^C$, $g(u)$ is a boolean value describing whether g is satisfied by u or not. When $g(u)$ is true, we shall say that u is a solution og g.

A *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.

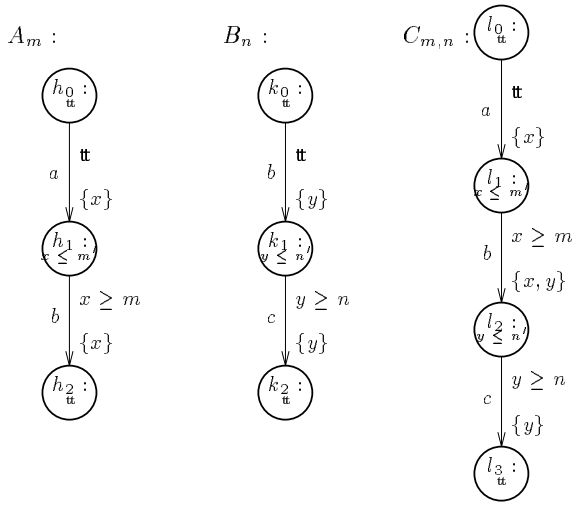


Fig. 1. Three timed automata

The semantics of A is 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 $I(u')$

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

Example 2. Reconsider the automaton $C_{m,n}$ of Figure 1. Assume that $d \geq 0$, $m \leq e \leq m'$ and $n \leq f \leq n'$. We have the following typical transition sequence:

$$(l_0, (0,0)) \xrightarrow{\epsilon(d)} (l_0, (d,d)) \xrightarrow{a} (l_1, (0,d)) \xrightarrow{\epsilon(e)} (l_1, (e, d+e)) \xrightarrow{b} (l_2, (0,0)) \xrightarrow{\epsilon(f)}$$
$$(l_2, (f,f)) \xrightarrow{c} (l_3, (f,0))$$

Note that we need to assume that $m \leq e \leq m'$ and $n \leq f \leq n'$ because of the invariant conditions on l_1 and l_2. □

Parallel composition may now be extended to timed automata in the obvious way: for two timed automata A and B and a synchronization function f, the parallel composition $A \mid_f B$ denotes the timed transition system $\mathcal{S}_A \mid_f \mathcal{S}_B$. Note that the timed transition system $\mathcal{S}_A \mid_f \mathcal{S}_B$ can also be represented finitely as a timed automaton. In fact, one may effectively construct the product automaton $A \otimes_f B$ such that its timed transition system $\mathcal{S}_{A \otimes_f B}$ is bisimilar to $\mathcal{S}_A \mid_f \mathcal{S}_B$. The nodes of $A \otimes_f B$ is simply the product of A's and B's nodes, the invariant conditions on the nodes of $A \otimes_f B$ are the conjunctions of the conditions on respective A's and B's nodes, the set of clocks is the (disjoint) union of A's

and B's clocks, and the edges are based on synchronizable A and B edges with enabling conditions conjuncted and reset–sets unioned.

Example 3. Let f be the synchronization function defined by $f(a,0) = a$, $f(b,b) = b$ and $f(0,c) = c$. Then the automaton $C_{m,n}$ in Figure 1 is timed bisimilar to the part of $A_m \otimes_f B_n$ which is reachable from (h_0, k_0). □

3 Timed Logics

We first introduce the syntax and semantics of the dense–time logic $\mathcal{L}_\nu$ presented in [17]. For the practical goal of verification of real–time systems, we find that it suffices to consider a certain fragment $\mathcal{L}_s$ especially designed to express safety and bounded liveness properties. Most importantly, as we shall show in subsequent sections, the rectriction to $\mathcal{L}_s$ allows for extremely efficient model–checking algorithms.

3.1 Syntax and Semantics

We first consider a dense–time logic $\mathcal{L}_\nu$ with clocks and recursion. This logic may be seen as a certain fragment [6] of the μ–calculus T_μ presented in [14]. In [17] it has been shown that this logic is sufficiently expressive that for any timed automaton one may construct a single *characteristic* formula uniquely characterizing the automaton up to timed bisimilarity. Also, decidability of a satisfiability [7] problem is demonstrated.

Definition 4. *Let K be a finite set of clocks. We shall call K formula clocks. Let Id be a set of identifiers. The set $\mathcal{L}_\nu$ of formulae over K, Id, Act, and $\mathcal{P}$ is generated by the abstract syntax with φ and ψ ranging over $\mathcal{L}_\nu$:*

$$\varphi ::= c \mid p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \exists\, \varphi \mid \forall\, \varphi \mid \langle a \rangle\, \varphi \mid [a]\, \varphi$$
$$\mid x \,\mathsf{in}\, \varphi \mid x + n \sim y + m \mid Z$$

where c is an atomic clock constraint in the form of $x \sim n$ or $x - y \sim n$ for $x, y \in K$ and natural number n, $p \in \mathcal{P}$ is an atomic predicate, $a \in Act$ is an action, $z \in K$ and $Z \in Id$ is an identifier. □

The meaning of the identifiers is specified by a declaration $\mathcal{D}$ assigning a formula of $\mathcal{L}_\nu$ 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, we interpret the $\mathcal{L}_\nu$ formulas over an extended state $\langle s, u \rangle$ where $s \in S$ is a state of $\mathcal{S}$, and u is a clock assignment for K. A formula

[6] allowing only maximal recursion and using a slightly different notion of model

[7] Bounded in the number of clocks and maximal constant allowed in the satisfying automata.

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

Table 1. Definition of satisfiability.

of the form: $x \sim m$ and $x - y \sim n$ is satisfied by an extended state $\langle s, u\rangle$ if the values of x, y in u satisfy the required relationship. Informally, an extended state $\langle s, u\rangle$ satisfies $\Forall\varphi$ means that all future states reachable from $\langle s, u\rangle$ by delays will satisfy property φ. Thus $\forall$ denotes universal quantification over delay transitions. Similarly, $\exists$ denotes existential quantification over delay transitions. A state $\langle s, u\rangle$ satisfies $[a]\varphi$ means that all intermediate states reachable from $\langle s, u\rangle$ by an a–transition (performed by s will satisfy property φ. Thus $[a]$ denotes universal quantification over a–transitions. Similarly, $\langle a\rangle$ denotes existential quantification over a–transitions. The formula $(x \,\mathsf{in}\, \varphi)$ initializes the formula clock x to 0; i.e. an extended state satisfies the formula in case the modified state with x being reset to 0 satisfies φ. Finally, an extended state satisfies an identifier Z if it satisfies the corresponding declaration (or definition) $\mathcal{D}(Z)$. 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. We have left out the cases for $\exists$ and $\langle a\rangle$ as they are immediate duals.

Any relation satisfying the implications in Table 1 is called a *satisfiability* relation. It follows from standard fixpoint theory [20] that $\models_{\mathcal{D}}$ is the union of all satisfiability relations. For simplicity, we shall omit the subscript $\mathcal{D}$ and write $\models$ instead of $\models_{\mathcal{D}}$ whenever it is understood from the context. We say that $\mathcal{S}$ satisfies a formula φ and write $\mathcal{S} \models \varphi$ when $\langle s_0, v_0\rangle \models \varphi$ where s_0 is the initial state of $\mathcal{S}$ and v_0 is the assignment with $v_0(x) = 0$ for all x. Similarly, we say that a timed automaton A satisfies φ in case $\mathcal{S}_A \models \varphi$. We write $A \models \varphi$ in this case.

Example 4. Consider the following declaration $\mathcal{F}$ of the identifiers X_i and Z_i where i is a natural number.

$$
\mathcal{F} \ = \ \Big\{ X_i \stackrel{\text{def}}{=} [a]\Big(z \,\mathsf{in}\, Z_i\Big) \quad , \quad Z_i \stackrel{\text{def}}{=} (\mathsf{at}(l_3) \vee \Big(z < i \wedge [a]Z_i \wedge [b]Z_i \wedge [c]Z_i \wedge \forall Z_i\Big)\Big) \Big\}
$$

Assume that $\mathsf{at}(l_3)$ is an atomic proposition meaning that the system is operating in control–node l_3. Then, X_i expresses the property that after an a–transition, the system must reach node l_3 within i time units. Now, reconsider the automata A_m, B_n and $C_{m,n}$ of Figure 1 and Examples 1 and 2. Then it may be argued that $C_{m,n} \models X_{m'+n'}$ and (consequently), that $A_m \mid_f B_n \models X_{m'+n'}$. $\square$

$$\mathsf{INV}(\varphi) \equiv X \text{ where } X \stackrel{\text{def}}{=} \varphi \wedge \Box X \wedge [Act]X$$
$$\varphi \ \mathsf{UNTIL} \ \psi \equiv X \text{ where } X \stackrel{\text{def}}{=} \psi \vee \Big(\varphi \wedge \Box X \wedge [Act]X\Big)$$
$$\varphi \ \mathsf{UNTIL}_{<n} \ \psi \equiv z \,\mathsf{in}\, \Big((\varphi \wedge z < n) \ \mathsf{UNTIL} \ \psi\Big)$$
$$\psi \ \mathsf{BEFORE} \ n \equiv \mathsf{tt} \ \mathsf{UNTIL}_{<n} \ \psi$$

Table 2. Derived Operators

3.2 Derived Operators

The property Z_i described in Example 3 is an attempt to specify bounded liveness properties: namely that a certain proposition must be satisfied within a given time bound. We shall use the more informative notation $\mathsf{at}(l_3)$ BEFORE i to denote Z_i. In the following, we shall present several such intuitive operators that are definable in our logic.

For simplicity, we shall assume that the set of actions Act is a finite set $\{a_1 ... a_m\}$, and use $[Act]\varphi$ to denote the formula $[a_1]\varphi \wedge ... \wedge [a_m]\varphi$. Now, let φ and ψ be a general formulas and n be a natural number. A collection of derived operators are given in Table 2.

The intuitive meanings of these operators are the following: $\mathsf{INV}(\varphi)$ is satisfied by a timed automaton provided φ holds in any reachable state; i.e. φ is an invariant property of the automaton. φ UNTIL ψ is satisfied by a timed automaton provided φ holds until the property ψ becomes true. Due to the maximal fixedpoint semantics this derived operator is the *weak* UNTIL–operator in that there is no guarantee that ψ ever becomes true. The bounded version of the UNTIL–construct φ $\mathsf{UNTIL}_{<n}$ ψ is similar to φ UNTIL ψ except that ψ must be true within n time units. A simpler version of this operator is ψ BEFORE n meaning that property ψ must be true within n time units.

3.3 A Logic for Safety and Bounded Liveness Properties

It has been pointed out [13, 22], that the practical goal of verification of real–time systems, is to verify simple safety properties such as deadlock–freeness and mutual exlusion. Similarly, we have found that for practical purposses it (often) suffices to use only a fragment of $\mathcal{L}_\nu$.

Formally, the logic for Safety and Bounded Liveness Properties, $\mathcal{L}_s$, is the fragment of $\mathcal{L}_\nu$ obtained by eliminating the use of the existential quantifiers $\exists$ (over delay transitions) and $\langle a \rangle$ (over a–transitions), and restricting the use of disjunction to formulas of the forms $c \vee \varphi$ (an atomic clock constrain) and $p \vee \varphi$ (an atomic proposition). The logic $\mathcal{L}_s$ is sufficiently expressive that we may specify a number of safety and bounded liveness properties. In particular, restricting ψ to c and p in Table 2 yields (restricted) derived operators expressible

in $\mathcal{L}_s$. Consequently the formulas of Example 4 are in $\mathcal{L}_s$. Most importantly, the restriction to $\mathcal{L}_s$ allows for extremely efficient automatic verification.

4 Region–Based Model–Checking

We have presented a model to describe real–time systems, i.e. networks of timed automata, and logics to specify properties of such systems. The next question is how to check whether a given logical formula is satisfied by a given network of automata. This is the so–called model–checking problem. The model-checking problem for $\mathcal{L}_\nu$ consists in deciding if a given timed automaton A satisfies a given specification φ in $\mathcal{L}_\nu$. This problem is decidable using the region technique of Alur, Courcoubetis and Dill [3, 2], which provides an abstract semantics of timed automata in the form of finite labelled transition systems with the truth value of $\mathcal{L}_\nu$ formulas being maintained.

The basic idea is that, given a timed automaton A, two states (l, u_1) and (l, u_2) which are close enough with respect to their clocks values (we will say that u_1 and u_2 are in the same *region*) can perform the same actions, and two extended states $\langle (l, u_1), v_1 \rangle$ and $\langle (l, u_2), v_2 \rangle$ where $u_1 v_1$ and $u_2 v_2$ are in the same region, satisfy the same $\mathcal{L}_\nu$–formulas. In fact the regions are defined as equivalence classes of a relation $\doteq$ over time assignments [14]. Formally, given C a set of clocks and k an integer, we say $v \doteq u$ if and only if v and u satisfy the same conditions of $\mathcal{B}_k(C)$. $[v]$ denotes the region which contains the time assignment v. $\mathcal{R}_k^C$ denotes the set of all regions for a set C of clocks and the maximal constant k. From a decision point of view it is important to note that $\mathcal{R}_k^C$ is finite.

For a region $\gamma \in \mathcal{R}_k^C$, we can define $b(\gamma)$ as the truth value of $b(v)$ for any v in γ. Conversely given a region γ, we can easily build a formula of $\mathcal{B}(C)$, called $\beta(\gamma)$, such that $\beta(\gamma)(v) = \mathtt{tt}$ iff $v \in \gamma$. Thus, given a region γ', $\beta(\gamma)(\gamma')$ is mapped to the value $\mathtt{tt}$ precisely when $\gamma = \gamma'$. Finally, note that $\beta(\gamma)$ itself can be viewed as a $\mathcal{L}_\nu$ formula.

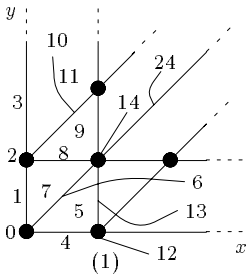
Given a region $[v]$ in $\mathcal{R}_k^C$ and $C' \subseteq C$ we define the following reset operator: $[C' \to 0][v] = [[C' \to 0]v]$. Moreover, for a region $[v]$, we define the successor region (denoted by $succ([v])$) as the region $[v']$, where:

$$v'(x) = \begin{cases} v(x) + f & \forall x \in C.\, v(x) > k \vee \{v(x)\} \neq 0 \\ v(x) + f/2 & \exists x \in C.\, v(x) \leq k \wedge \{v(x)\} = 0 \end{cases}$$

where $f = min\{1 - \{v(x)\} \quad | \quad v(x) \leq k\}$ [8]. Informally the change from γ to $succ(\gamma)$ correspond to the minimal elapse of time which can modify the enabled actions of the current state.

We denote by γ^i the i^{th} successor region of γ (i.e. $\gamma^i = succ^i(\gamma)$). From each region γ, it is possible to reach a region γ' s.t. $succ(\gamma') = \gamma'$, and we denote by i_γ the required number of step s.t. $\gamma^{i_\gamma} = succ(\gamma^{i_\gamma})$.

[8] if this set is empty, then $f = 0$



(1)

$succ(\gamma_0) = \gamma_6$
$succ(\gamma_5) = \gamma_{13}$
$\ldots$

$\beta(\gamma_6) = \quad 0 < x < 1 \wedge\ 0 < y < 1 \wedge\ y = x$

$\beta(\gamma_0) = (x = 0 \wedge y = 0)$
$\beta(\gamma_8) = (0 < x < 1 \wedge y = 1)$

$\beta(\gamma_7) = \quad 0 < x < 1 \wedge\ 0 < y < 1 \wedge\ y > x$

Fig. 2. $\mathcal{R}_k^C$ with $C = \{x, y\}$ and $k = 1$

Example 5. The Figure 2 gives an overview of the set of regions defined by two clocks x and y, and the maximal constant 1. In this case there are 32 different regions. In general successor regions are determined by following 45^o lines upwards to the right. □

Let $A = \langle N, l_0, E, I, V \rangle$ be a timed automaton over actions Act, atomic propostions $\mathcal{P}$ and clocks C. Let k_A denotes the maximal constant occurring in the enabling condition of the edges E. Then for any $k \geq k_A$ we can now define a region–based semantics of A over region–states $[l, \gamma]$ where $l \in N$ and $\gamma \in \mathcal{R}_k^C$ as follows: for any $[l, \gamma]$ we have $[l, \gamma] \xrightarrow{a} [l', \gamma']$ iff $\exists\ v \in \gamma,\ \ (l, v) \xrightarrow{a} (l', v')$ and $v' \in \gamma'$.

Consider now $\mathcal{L}_\nu$ with respect to formula clock set K and maximal constant k_L (assuming that K and C are disjoint). Then an *extended region–state* is a pair $[l, \gamma]$ where $l \in N$ and $\gamma \in \mathcal{R}_k^{C \cup K}$ with $k = max(k_A, k_L)$. We define now the *region–based semantics* for $\mathcal{L}_\nu$, i.e. the truth value of $\mathcal{L}_\nu$ formulas over extended region–states. Formally, $\vdash_D$ is the largest relation satisfying the implications of Table 3[9]. We have left out the cases for $\exists$ and $\langle a \rangle$ as they are immediate duals. Also, when no confusion can occur we omit the subscript and write $\vdash$ instead of $\vdash_D$. This symbolic interpretation of $\mathcal{L}_\nu$ is closely related to the standard interpretation as stated by the following important result: Let φ be a formula of $\mathcal{L}_\nu$, and let $\langle (l, u), v \rangle$ be an extended state over some timed automaton A, then we have

$$\langle (l, u), v \rangle \models \varphi \quad \text{if and only if} \quad [l, [uv]] \vdash \varphi$$

It follows that the model checking problem for $\mathcal{L}_\nu$ is decidable since it suffices to check the truth value of any given $\mathcal{L}_\nu$ formula φ with respect to the finite transition system corresponding to the extended region–state semantics of A.

[9] $\gamma_{|C}$ (resp. $\gamma_{|K}$) denotes the set of time-assignments in γ restricted to the automata (resp. formula) clocks.

$$
\begin{array}{ll}
[l,\gamma] \vdash c & \Rightarrow c(\gamma) \\
[l,\gamma] \vdash p & \Rightarrow p \in V(l) \\
[l,\gamma] \vdash \varphi \wedge \psi & \Rightarrow [l,\gamma] \vdash \varphi \text{ and } [l,\gamma] \vdash \psi \\
[l,\gamma] \vdash \varphi \vee \psi & \Rightarrow [l,\gamma] \vdash \varphi \text{ or } [l,\gamma] \vdash \psi \\
[l,\gamma] \vdash \mathbb{V}\varphi & \Rightarrow \forall i \in \mathbf{N}.\ \ [l, succ^i(\gamma)] \vdash \varphi \\
[l,\gamma] \vdash [a]\varphi & \Rightarrow \forall\, [l',\gamma'].\ \ [l,\gamma_{|C}] \xrightarrow{a} [l',\gamma'_{|C}] \text{ and } \gamma'_{|K} = \gamma_{|K} \\
 & \qquad\qquad \text{implies } [l',\gamma'] \vdash \varphi \\
[l,\gamma] \vdash x \,\mathsf{in}\, \varphi & \Rightarrow [l,[\{x\} \to 0]\gamma] \vdash \varphi \\
[l,\gamma] \vdash Z & \Rightarrow [l,\gamma] \vdash D(Z)
\end{array}
$$

Table 3. Definition of region–based satisfiablity.

5 Symbolic Model–Checking

The region–graph technique applied in the previous section allows the state space of a real time system to be partitioned into finitely many regions in such a way that states within the same region satisfy the same properties. However, as the notion of region is essentially property–independent and the number of such regions depends highly on the constants used in the clock constraints of an automaton, the region partitioning is extremely fine (and large). In this section we shall offer a much coarser (and smaller) partitioning of the state space yielding extremely efficient model–checking for the safety logic $\mathcal{L}_s$.

Recall that a semantical state of a network of timed automata is a pair (l,u) where l is a control-node and $u \in \mathbf{R}^C$ is a clock assignment. The model-checking problem is in general to check whether an extended state in the form $\langle (l,u), v\rangle$ satisfy a given formula φ, that is,

$$\langle (l,u), v\rangle \models \varphi$$

Note that u is a clock assignment for the automata clocks and v is a clock assignment for the formula clocks. Now, the problem is that we have too many (in fact, infinitely many) such assignments to check in order to conclude $\langle (l,u), v\rangle \models \varphi$.

In this section, we shall use clock constraints $\mathcal{B}(C \cup K)$ for automata clocks C and formula clocks K, as defined in section 2 to symbolically represent clock assignments. We shall use D to range over $\mathcal{B}(C \cup K)$. For safety formulas $\varphi \in \mathcal{L}_s$ we develop an algorithm to simultaneously check

$$[l, D] \models \varphi$$

which means that for each u and v such that uv is a solution to the constraint system D, we have $\langle (l,u), v\rangle \models \varphi$.

Thus the space $\mathbf{R}^{C \cup K}$ is partitioned in terms of clock constraints. As for a given network and a given formula, we have only finite many such constraints to check, the problem becomes decidable, and in fact as the partitioning

takes account of the particular property, the number of partitions is in practice considerably smaller compared with the region-technique.

5.1 Operations on Clock Constraints

To develop the model-checking algorithm, we need a few operations to manipulate clock constraints. Given a clock constraint D, we shall call the set of clock assignments satisfying D, the *solution set* of D.

Definition 5. *Let A and A' be the solution sets of clock constraints $D, D' \in \mathcal{B}(C \cup K)$. We define*

$$\begin{aligned} A^{\uparrow} &= \{w + d \mid w \in A \text{ and } d \in \mathbf{R}\} \\ A^{\downarrow} &= \{w \mid \exists d \in \mathbf{R} : \ w + d \in A\} \\ \{x\}A &= \{[\{x\} \mapsto 0]w \mid w \in A\} \\ A \wedge A' &= \{w \mid w \in A \text{ and } w \in A'\} \end{aligned}$$

□

First, note that $A \wedge A'$ is simply the intersection of the two sets. Consider the set A for the case of two clocks, shown in (a) of Figure 3. The three operations $A^{\uparrow}$, $A^{\downarrow}$ and $\{x\}A$ are illustrated in (b), (c) and (d) respectively of Figure 3. Intuitively, $A^{\uparrow}$ is the largest set of time assignments that will eventually reach A after some delay; whereas $A^{\downarrow}$ is the dual of $A^{\uparrow}$: namely that it is the largest set of time assignments that can be reached by some delay from A. Finally, $\{y\}A$ is the projection of A down to the x-axis. We extend the projection operator to sets of clocks. Let $r = \{x_1 ... x_n\}$ be a set of clocks. We define $r(A)$ recursively by $\{\}(A) = A$ and $\{x_1 ... x_n\}(A) = \{x_1\}(\{x_2 ... x_n\}A)$.

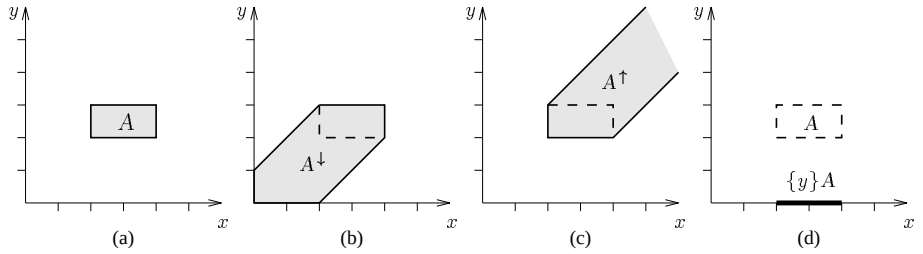


Fig. 3. Operations on Solution Sets

The following Proposition establishes that the class of clock constraints $\mathcal{B}(C \cup K)$ is closed under the four operations defined above.

Proposition 6. *Let $D, D' \in \mathcal{B}(C \cup K)$ with solution sets A and A', and $x \in C \cup K$. Then there exist $D_1, D_2, D_3, D_4 \in \mathcal{B}(C \cup K)$ with solution sets $A^{\uparrow}$, $A^{\downarrow}$, $\{x\}A$ and $A \wedge A'$ respectively.* □

In fact, the resulted constraints D_i's can be effectively constructed from D and D', as shown in section 4.3. In order to save notation, from now on, we shall simply use $D^{\uparrow}$, $D^{\downarrow}$, $\{x\}D$ and $D \wedge D'$ to denote the clock constraints which are guaranteed to exist due to the above proposition. We will also need a few *predicates* over clock constraints for the model–checking procedure. We write $D \subseteq D'$ to mean that the solution set of D is included in the solution set of D' and $D = \emptyset$ to mean that the solution set of D is empty.

5.2 Model–Checking by Constraint Solving

Given a network of timed automaton A over clocks C, we shall interprete formulas of $\mathcal{L}_s$ over clocks K with respect to symbolic states of the form $[l, D]$ where l is a control-node of A and D is a clock constraint of $\mathcal{B}(C \cup K)$. Let $\mathcal{D}$ be a declaration. The symbolic satisfaction relation $\vdash_{\mathcal{D}}$ between symbolic states and formulas of $\mathcal{L}_s$ is defined as the largest relation satisfying the implications in Table 4. We call a relation satisfying the implications in Table 4 a *symbolic*

$$
\begin{array}{lll}
D = \emptyset & \Rightarrow & [l, D] \vdash \varphi \\
[l, D] \vdash c & \Rightarrow & D \subseteq c \\
[l, D] \vdash p & \Rightarrow & p \in V(s) \\
[l, D] \vdash c \vee \varphi & \Rightarrow & [l, D] \vdash [l, D \wedge \neg c] \vdash \varphi \\
[l, D] \vdash p \vee \varphi & \Rightarrow & [l, D] \vdash p \text{ or } [l, D] \vdash \varphi \\
[l, D] \vdash \varphi_1 \wedge \varphi_2 & \Rightarrow & [l, D] \vdash \varphi_1 \text{ and } [l, D] \vdash \varphi_2 \\
[l, D] \vdash [a]\,\varphi & \Rightarrow & [l', r(D \wedge g)] \vdash \varphi \text{ whenever } l \xrightarrow{g,a,r} l' \\
[l, D] \vdash \forall\!\!\forall \varphi & \Rightarrow & [l, D] \vdash \varphi \text{ and } [l, (D \wedge I(n))^{\uparrow} \wedge I(l)] \vdash \varphi \\
[l, D] \vdash x \text{ in } \varphi & \Rightarrow & [l, \{x\}D] \vdash \varphi \\
[l, D] \vdash Z & \Rightarrow & [l, D] \vdash \mathcal{D}(Z)
\end{array}
$$

Table 4. Definition of symbolic satisfiability.

satisfiability relation. Again, it follows from standard fixpoint theory [20] that $\vdash_{\mathcal{D}}$ is the union of all symbolic satisfiability relations. For simplicity, we shall omit the index $\mathcal{D}$ and write $\vdash_{\mathcal{D}}$ instead of $\vdash$ whenever it is understood from the context.

The following Theorem shows that the symbolic interpretation of $\mathcal{L}_s$ in Table 4 expresses the sufficient and necessary conditions for a timed automata to satisfy a formula φ [10].

[10] Note that Theorem cannot be extended to a logic with general disjunction (or existential quantifications): the obvious requirement that $[l, D] \models \varphi_1 \vee \varphi_2$ should imply either $[l, D] \models \varphi_1$ or $[l, D] \models \varphi_2$ will fail to satisfy the Theorem.

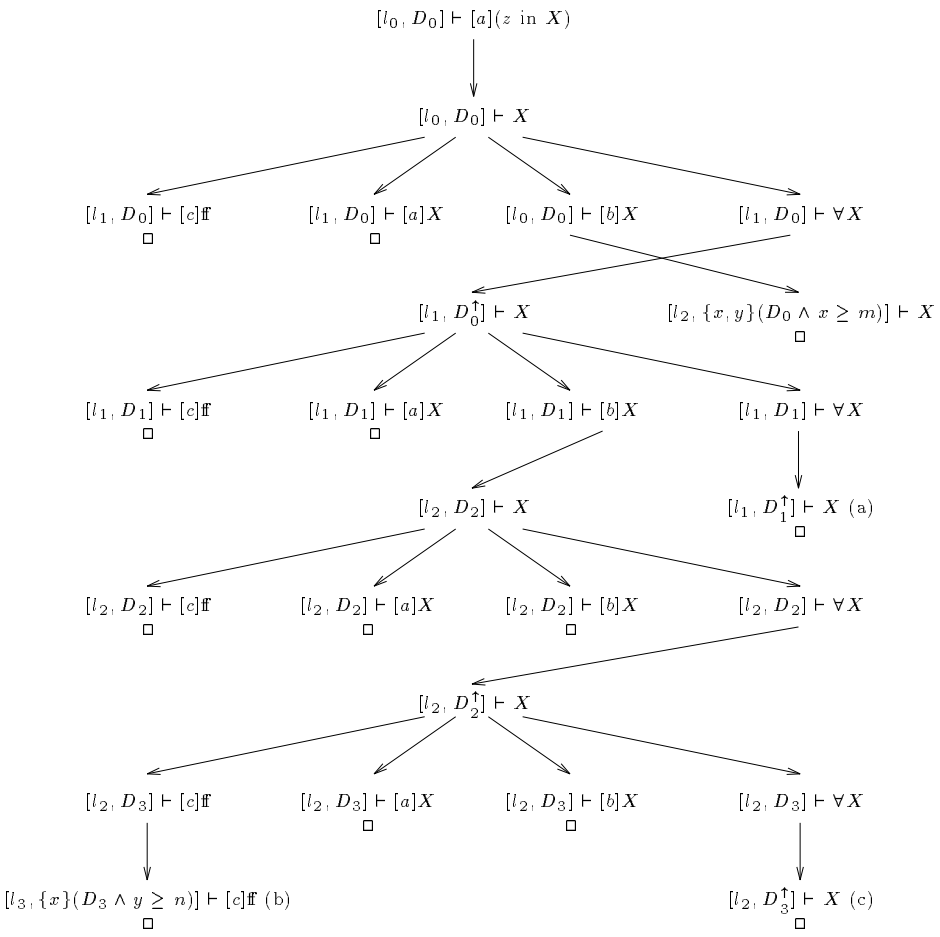


Fig. 4. Rewrite Tree of $[l_0, D_0] \vdash [a](z \text{ in } X)$.

Theorem 7. *Let A be a timed automaton over clock set C and φ a formula of $\mathcal{L}_s$ over K. Then the following holds:*

$$A \models \varphi \quad \textit{if and only if} \quad [l_0, D_0] \vdash \varphi$$

where l_0 is the initial node of A and D_0 is the linear constraint system $\{x = 0 \mid x \in C \cup K\}$. □

Given a symbolic satisfaction *problem* $[l, D] \vdash \varphi$ we may determine its validity by using the implications of Table 4 as rewrite rules. Due to the maximal fixed point property of $\vdash$, rewriting may be terminated successfully in case cycles are encountered. As the rewrite graph of any given problem $[l, D] \vdash \varphi$ can be shown to be finite this yields a decision procedure for model checking.

Example 6. Reconsider the automaton $C_{m,n}$ in Figure 1 assuming that $m' = n' = +\infty$ (making the invariants of l_1 and l_2 true). Consider the property $(z \,\mathsf{in}\, X)$ where X is defined as follows:

$$X \stackrel{\text{def}}{=} (z \geq i) \vee ([c]\mathbf{ff} \wedge [a]X \wedge [b]X \wedge \forall X)$$

The property $(z \,\mathsf{in}\, X)$ expresses that the accumulated time between an initial a-action and a following c-action must exceed i. We want to show that $C_{m,n}$ satisfies this property provided the sum of the delays m and n exceeds the required delay i. That is, we must show $[l_0, D_0] \vdash [a](z \,\mathsf{in}\, X)$ provided $n + m \geq i$. The generated rewrite tree (i.e. execution tree of our model checking procedure) is illustrated in Figure 4. The constraints used are the following:

$$\begin{array}{ll}
D_0 = \{x = y = z = 0\} & D_2 = \{x, y\}(D_1 \wedge x \geq m) \equiv \{x = y = 0, m \leq z < i\} \\
D_0^{\uparrow} = \{x = y = z\} & D_2^{\uparrow} = \{x = y, m \leq z - x < i\} \\
D_1 = D_0^{\uparrow} \wedge (z < i) & D_3 = D_2^{\uparrow} \wedge z < i = \{x = y, m \leq z - x < i, z < i\} \\
\quad \equiv \{x = y = z, z < i\} & D_3^{\uparrow} = D_2^{\uparrow} \\
D_1^{\uparrow} = D_0^{\uparrow} &
\end{array}$$

In the rewrite tree a node (i.e. a problem) is related to its sons by application of the appropriate rewrite rule of Table 4: i.e. the sons represent the conjuncts of the right-hand side of the applied rule [11]. The leaves of the tree are either obviously valid problems or reoccurrences. The leaf-problem labeled (b) is valid as $(D_3 \wedge y \geq n) = \emptyset$ holds under the assumption that $n + m \geq i$. Thus (b) is an instance of the first rule of Table 4. The problem labeled (a) is a reoccurrence of the earlier problem $[l_1, D_0{}^{\uparrow}]$ as it can be shown that $D_0{}^{\uparrow} = D_1{}^{\uparrow}$. Similarly, (c) is a reoccurrence. □

5.3 Implementation Issues

The operations and predicates on clock constraint systems discussed in Section 5.1 can be efficiently implemented by representing constraint systems as weighted directed graphs. The basic idea is to use a shortest–path algorithm to close a constraint system under entailment so that operations and predicates can be easily computed.

Given a clock constraint system D over a clock set C, we represent D as a weighted directed graph with vertices $C \cup \{0\}$. The graph will have an edge from x to y with weight m provided $x - y \leq m$ is a constraint of D. Similarly, there will be an edge from 0 to x (from x to 0) with weight m whenever $x \leq m$ ($x \geq -m$) is a constraint of D [12].

A clock constraint system D is *closed under entailment* if no constraint of D can be strengthened without reducing the solution set. For closed constraint systems D and D' the inclusion and emptiness predicates are easy to decide:

[11] For problems involving an identifier, the tree reflects *two* successive rule applications starting with the unfolding of the identifier.

[12] In this presentation we have made the simplifying assumption that D does not contain any strict constraints, i.e. constraints of the form $x - y < n$.

$D \subseteq D'$ holds iff for any constraint in D' there is a tighter constraint in D (e.g. whenever $(x - y \leq m) \in D'$ then $(x - y \leq m') \in D$ for some $m' \leq m$); $D = \emptyset$ holds if D contains two contradicting constraints (e.g. $x - y \leq m$ and $x - y \geq n$ where $m < n$). To close a clock constraint system D amounts to solve the shortest-path problem for its graph and can thus be computed in $\mathcal{O}(n^3)$ (which is also the complexity for the inclusion and emptiness predicates), where n is the number of clocks.

Given constraint systems D and D' the operations $D^{\uparrow}$, $D^{\downarrow}$, $\{x\}D$ and $D \wedge D'$ can be computed in $\mathcal{O}(n^2)$. The complexity of the operation $c \wedge D$, where c is an atomic constraint, is $\mathcal{O}(1)$.

6 Compositional Model–Checking

The symbolic model–checking presented in the previous section provides an efficient way to deal with the potential explosion caused by the addition of clocks. However, a potential explosion in the node–space due to parallel composition still remains. In this section we attack this problem by development of a quotient construction, which allows components to be gradually moved from the parallel system into the specification, thus avoiding explicit construction of the global node space. The intermediate specifications are kept small using minimization heuristics. Recent experimental work by Andersen [4] demonstrates that for (untimed) finite–state systems the quotient technique improves results obtained using Binary Decision Diagrams. Also, an initial experimental investigation of the quotient technique to real–time systems in [18] has indicated that these promising results will carry over to the setting of real–time systems. In this section we shall provide a new (and compared with [18] simple) quotient construction and show how to integrate it with the symbolic technique of the previous section.

6.1 Quotient Construction

Given a formula φ of $\mathcal{L}_\nu$, and two timed automata A and B, we aim at constructing a formula (called the *quotient*) $\varphi /_f B$ such that

$$A \mid_f B \models \varphi \quad \text{if and only if} \quad A \models \varphi /_f B \tag{3}$$

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 $A \mid_f B$. 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

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

Table 5. Definition of Quotient $\varphi\big/_f\, l$

the quotient formula $\varphi\big/_f\, l$ over $C \cup K$ in Table 5 on the structure of φ [13] [14]. We have left out the case for $\langle a \rangle$ as it is dual to that of $[a]$.

The quotient $\varphi\big/_f\, l$ expresses the sufficient and necessary requirement to a timed automaton A in order that the parallel composition $A \,|_f\, B$ with B at node l satisfies φ. In most cases quotienting simply distributes with respect to the formula construction. The quotient construction for $\forall\!\!\forall\varphi$ reflects that $A \,|_f\, B$ can only delay provided $I(l)$ is satisfied. The quotient construction for $[a]\varphi$ must quantify over all actions of A which can possibly lead to an a–transition of $A \,|_f\, B$: according to the semantics of parallel composition, b is such an action provided B (at node l) can perform a synchronizable action c (according to some edge $l \xrightarrow{g,c,r} l'$) such that $f(b,c) = a$. The guard as well as the reset set of the involved A-edge $l \xrightarrow{g,c,r} l'$ is reflected in the quotient formula.

Note that the quotient construction for identifiers introduces new identifiers of the form X_l. These new identifiers and their definitions are collected in the (quotient) declaration $\mathcal{D}_B$.

For l_0 the initial node of a timed automaton B, the quotient $\varphi\big/_f\, l_0$ ex-

[13] 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.

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

presses the sufficient and necessary requirement to a timed automaton A in order that the parallel composition $A \mid_f B$ satisfies φ. More precisely:

Theorem 8. *Let A and B be two timed automata and let l_0 be the initial node of B. Then*

$$A \mid_f B \models_{\mathcal{D}} \varphi \qquad \textit{if and only if} \qquad A \models_{\mathcal{D}_B} \left(\varphi /_f l_0\right)$$

□

Example 7. Reconsider the network, synchronization function and property from Examples 1, 2, 3 and 6. We want to establish that the network $A_m \mid_f B_n$ satisfies the following property Y provided $n + m \geq i$:

$$Y \stackrel{\text{def}}{=} [a]\Big(z \text{ in } X\Big) \qquad X \stackrel{\text{def}}{=} (z \geq i) \vee \Big([c]\mathbf{ff} \wedge [a]X \wedge [b]X \wedge \mathbb{W}X\Big)$$

From Theorem 8 it follows that the sufficient and necessary requirement to A_m in order that $A_m \mid_f B_n$ satisfies Y is that A_m satisfies $Y /_f k_0$. Using the quotient definition from Table 5 we get:

$$\begin{aligned}
Y /_f k_0 &\stackrel{\text{def}}{=} z \text{ in } (X /_f k_0) \\
X /_f k_0 &\stackrel{\text{def}}{=} (z \geq i) \vee \Big([b](y \text{ in } X /_f k_1) \wedge \mathbb{W}(X /_f k_0)\Big) \\
X /_f k_1 &\stackrel{\text{def}}{=} (z \geq i) \vee \Big((y \geq n \Rightarrow [c]\mathbf{ff}) \wedge \mathbb{W}(X /_f k_1)\Big)
\end{aligned}$$

□

6.2 Minimizations

It is obvious that repeated quotienting leads to an explosion in the formula. 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. In UPPAAL an "on-the-fly" technique insures 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: $\mathbf{ff} \wedge \varphi \equiv \mathbf{ff}$, $\mathbf{tt} \wedge \varphi \equiv \varphi$, $\langle a \rangle \mathbf{ff} \equiv \mathbf{ff}$, $\exists \mathbf{ff} \equiv \mathbf{ff}$, $x \text{in} \mathbf{ff} \equiv \mathbf{ff}$, $\langle a \rangle \varphi \wedge [a]\mathbf{ff} \equiv \mathbf{ff}$.

Constraint Propagation: Constraints on formula clocks may be propagated using various distribution laws (see Table 6). In some cases, propagation will

$$
\begin{aligned}
\emptyset \Rightarrow \varphi &\equiv \mathtt{tt} \\
D \Rightarrow c &\equiv \mathtt{tt} \quad ; \text{ if } D \subseteq c \\
D \Rightarrow ([a]\varphi) &\equiv [a](D \Rightarrow \varphi) \\
D \Rightarrow (\varphi_1 \wedge \varphi_2) &\equiv (D \Rightarrow \varphi_1) \wedge (D \Rightarrow \varphi_2) \\
D \Rightarrow (x \,\mathsf{in}\, \varphi) &\equiv x \,\mathsf{in}\, (\{x\}D \Rightarrow \varphi) \\
D \Rightarrow (p \vee \varphi) &\equiv p \vee (D \Rightarrow \varphi) \\
D \Rightarrow (c \vee \varphi) &\equiv (D \wedge \neg c) \Rightarrow \varphi \\
D \Rightarrow (\mathbb{V}\varphi) &\equiv \mathbb{V}(D^{\uparrow} \Rightarrow \varphi) \quad ; \text{ if } D^{\downarrow} \subseteq D \\
D \Rightarrow X &\equiv D \Rightarrow \mathcal{D}(X)
\end{aligned}
$$

Table 6. Constraint Propagation

$$
\begin{aligned}
\beta(\gamma) \Rightarrow c &\equiv \begin{cases} \mathtt{tt} & ; c(\gamma) \\ \beta(\gamma) \Rightarrow \mathtt{ff} & ; \text{otherwise} \end{cases} \\
\beta(\gamma) \Rightarrow (\mathbb{V}\varphi) &\equiv \mathbb{V}\Big(\bigwedge_{i=0\ldots i_\gamma} \beta(\gamma^i) \Rightarrow \varphi \Big) \\
\beta(\gamma) \Rightarrow (\varphi_1 \vee \varphi_2) &\equiv (\beta(\gamma) \Rightarrow \varphi_1) \vee (\beta(\gamma) \Rightarrow \varphi_2)
\end{aligned}
$$

Table 7. Region Propagation

lead to trivial clock constraints, which may be simplified to either $\mathtt{tt}$ or $\mathtt{ff}$ and hence made applicable to Boolean Simplification.

Region Propagation: For constraint identifying single regions, i.e. constraints of the form $\beta(\gamma)$ additional distribution laws are given in Table 7

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 $\varphi[\mathtt{tt}/S]$ [15] 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.

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

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 [16]. 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.

In the following Examples we apply the above transformation strategies to the quotient formula obtained in Example 7. In particular, the strategies will find the quotient formula to be trivially true in certain cases.

Example 8. Reconsider Example 7 with Y_0, X_0 and X_1 abbreviating $Y \big/_f k_0$, $X \big/_f k_0$ and $X \big/_f k_1$. Now Y_0 is the sufficient and necessary requirement to A_m in order that $A_m \,|_f\, B_n$ satisfies Y. From the definition of satisfiability for timed automata we see that:

$$A_m \models Y_0 \quad \text{if and only if} \quad A_m \models \mathtt{tt} \Rightarrow \Big(y \,\mathsf{in}\, Y_0\Big)$$

This provides an initial basis for constraint propagation. Using the propagation laws from Table 6 we get:

$$\mathtt{tt} \Rightarrow \Big(y \,\mathsf{in}\, Y_0\Big) \;\equiv\; \mathtt{tt} \Rightarrow \Big(\{y,z\} \,\mathsf{in}\, X_0\Big) \;\equiv\; \{y,z\} \,\mathsf{in}\, \Big(D_0 \Rightarrow X_0\Big)$$

where $D_0 = (y = 0 \wedge z = 0)$. This makes the implication $D_0 \Rightarrow X_0$ applicable to constraint propagation as follows:

$$\begin{aligned}
(D_0 \Rightarrow X_0) &\equiv D_0 \Rightarrow \Big[(z \geq i) \vee \Big([b](y \,\mathsf{in}\, X_1) \wedge \Forall X_0\Big)\Big] \\
&\equiv \Big(D_0 \Rightarrow [b](y \,\mathsf{in}\, X_1)\Big) \;\wedge\; \Big(D_0 \Rightarrow \Forall X_0\Big) \qquad \text{as } (z < i \wedge D_0) = D_0 \\
&\equiv [b]\Big(y \,\mathsf{in}\, (D_0 \Rightarrow X_1)\Big) \;\wedge\; \Forall\Big({D_0}^{\uparrow} \Rightarrow X_0\Big)
\end{aligned}$$

Continuing constraint propagation yields the equations in Table 8, where $D_1 = (y = 0 \wedge z < i)$. □

Example 9. (Example 8 Continued) Now consider the case when $n \geq i$. That is the delay n of the component B_n exceeds the delay i required as a minimum by the property Y. Thus the component B_n ensures on its own the satisfiability of Y; i.e. for *any* choice of A the system $A \,|_f\, B_n$ will satisfy Y. In

[16] For the full logic T_μ the equivalence problem is undecidable.

$$
\begin{aligned}
(D_0 \Rightarrow X_0) &\equiv [b]\big(y \,\mathsf{in}\, (D_0 \Rightarrow X_1)\big) \wedge \mathbb{W}\big({D_0}^{\uparrow} \Rightarrow X_0\big)\\
({D_0}^{\uparrow} \Rightarrow X_0) &\equiv [b]\big(y \,\mathsf{in}\, (D_1 \Rightarrow X_1)\big) \wedge \mathbb{W}\big({D_0}^{\uparrow} \Rightarrow X_0\big)\\
(D_1 \Rightarrow X_1) &\equiv \big((D_1 \wedge y \geq n) \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}({D_1}^{\uparrow} \Rightarrow X_1)\\
(D_0 \Rightarrow X_1) &\equiv \big((D_0 \wedge y \geq n) \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}({D_0}^{\uparrow} \Rightarrow X_1)\\
({D_0}^{\uparrow} \Rightarrow X_1) &\equiv \big(({D_0}^{\uparrow} \wedge z < i \wedge y \geq n) \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}(({D_0}^{\uparrow} \wedge z < i)^{\uparrow} \Rightarrow X_1)\\
({D_1}^{\uparrow} \Rightarrow X_1) &\equiv \big(({D_1}^{\uparrow} \wedge z < i \wedge y \geq n) \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}(({D_1}^{\uparrow} \wedge z < i)^{\uparrow} \Rightarrow X_1)
\end{aligned}
$$

Table 8. Equations after Constraint Propagation

$$
\begin{aligned}
(D_0 \Rightarrow X_0) &\equiv [b]\big(y \,\mathsf{in}\, (D_0 \Rightarrow X_1)\big) \wedge \mathbb{W}\big({D_0}^{\uparrow} \Rightarrow X_0\big)\\
({D_0}^{\uparrow} \Rightarrow X_0) &\equiv [b]\big(y \,\mathsf{in}\, (D_1 \Rightarrow X_1)\big) \wedge \mathbb{W}\big({D_0}^{\uparrow} \Rightarrow X_0\big)\\
(D_1 \Rightarrow X_1) &\equiv \big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}({D_1}^{\uparrow} \Rightarrow X_1)\\
(D_0 \Rightarrow X_1) &\equiv \big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}({D_0}^{\uparrow} \Rightarrow X_1)\\
({D_0}^{\uparrow} \Rightarrow X_1) &\equiv \big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}({D_0}^{\uparrow} \Rightarrow X_1)\\
({D_1}^{\uparrow} \Rightarrow X_1) &\equiv \big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\big) \wedge \mathbb{W}({D_1}^{\uparrow} \Rightarrow X_1)
\end{aligned}
$$

Table 9. Equations after Simplification

this particular case (i.e. $n \geq i$) it is easy to see that $({D_i}^{\uparrow} \wedge z < i \wedge y \geq n) = \mathtt{ff}$ for $i = 0, 1$ as ${D_i}^{\uparrow}$ ensures $z \geq y$. Also for $i = 0, 1$, $(D_i \wedge y \geq n) = \mathtt{ff}$ as $D_i \Rightarrow y = 0$ and we assume $n > 0$. Finally, it is easily seen that $({D_i}^{\uparrow} \wedge z < i)^{\uparrow} = {D_i}^{\uparrow}$ for $i = 0, 1$. Inserting these observations — which all may be efficiently computed — in the equations of Table 8 we get the simplified equations in Table 9. Now, the conjuncts $\mathtt{ff} \Rightarrow [c]\mathtt{ff}$ are obviously equivalent to $\mathtt{tt}$ and will thus be removed by the boolean simplification transformations. Now, using our strategy for Trivial Equation Elimination, it may be found that all the equations in Table 9 are trivial and may consequently be removed (simplified to $\mathtt{tt}$). To see this, simply observe that substituting $\mathtt{tt}$ for $D_i \Rightarrow X_j$ and ${D_i}^{\uparrow} \Rightarrow X_j$ on all right-hand sides in Table 9 leads to formulas which clearly can be simplified to $\mathtt{tt}$. Thus, in the case $n \geq i$, our minimization heuristics will yield $\mathtt{tt}$ as the property required of A in order that $A \mid_f B_n$ satisfies Y. □

7 Experimental Results

The techniques presented in previous sections have been implemented in our verification tool UPPAAL in C++. We have tested UPPAAL by various examples. We

also perform experiments on three existing real–time verification tools: HyTech (Cornell), Kronos (Grenoble), and Epsilon (Aalborg). Though the compositional model–checking technique is still under implementation, our experimental results show that UPPAAL is not only substantially faster than the other tools but also able to handle much larger systems.

In particular, we have used Fisher's mutual exclusion protocol in our experiments on the tools. 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 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.

7.1 Fischer's Mutual Exclusion Protocol

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.

Assume a concurrent system with n processes $\mathsf{P}_1 ... \mathsf{P}_n$. Each process P_i with i being its identifier, has a clock x_i. We model the shared variable as a timed automaton V over the set of atomic actions: $\mathsf{A_V} = \{v := i \mid i = 0...n\} \cup \{v = i \mid i = 0...n\}$, and $\mathsf{V} = \langle N, h_0, E, I, V \rangle$ where $N = \{\mathsf{V}_0 ... \mathsf{V}_n\}$, $h_0 = \mathsf{V}_0$, $E = \{\langle \mathsf{V}_i, \mathtt{tt}, v := j, \emptyset, \mathsf{V}_j \rangle \mid i, j = 0...n\} \cup \{\langle \mathsf{V}_i, \mathtt{tt}, v = i, \emptyset, \mathsf{V}_i \rangle \mid i = 0...n\}$, I is defined by $I(\mathsf{V}_i) = \mathtt{tt}$ for all $i \leq n$ and we simply assume V is defined by $V(\mathsf{V}_i) = \emptyset$ for all $i \leq n$.

The automaton for a typical process P_i is shown in Fig 5. We assume that the invariant conditions on nodes are all $\mathtt{tt}$ in this particular example. Moreover, we assume that the proposition assignment function is defined in such a way that $\mathsf{at}(l') \in V(l)$ if $l' = l$ and $\neg\mathsf{at}(l') \in V(l)$ if $l' \neq l$ for all nodes l and l'. Note that in the clock constraints $x_i < m_1$ and $x_i > m_2$, we have used two parameters. They can be any natural numbers satisfying the condition $m_1 \leq m_2$. Now, the whole protocol is described as the following network:

$$\mathsf{FISCHER}_n \equiv (\mathsf{P}_1 | \mathsf{P}_2 | ... | \mathsf{P}_n) || \mathsf{V}$$

where $|$ and $||$ are the full interleaving and full syncronization operators, induced by synchronization functions f and g respectively, defined by $f(0, a) = a$, $f(a, 0) = a$, and $g(a, a) = a$.

This is a simplified version of the original protocol and has been studied in e.g. [1, 19], which permits only one process to enter the critical section and never exits it. Recovery actions from failure to enter the critical section are omitted. However, it can be easily extended to model the full version of the protocol.

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

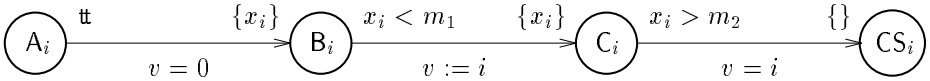


Fig. 5. Fischer's Mutual Exclusion Protocol

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:

$$\mathsf{MUTEX}_{1,2} \equiv \mathsf{INV}\Big(\neg \mathsf{at}(\mathsf{CS}_1) \vee \neg \mathsf{at}(\mathsf{CS}_2)\Big)$$

So we need to prove the theorem

$$\mathsf{FISCHER}_n \models \mathsf{MUTEX}_{1,2}$$

7.2 Performance Evaluation

Using the current version of our tool UPPAAL , installed on a SparcStation 10 running SunOS 4.1.2 with 64MB of primary memory and 64MB of swap memory, we have verified the mutual exclusion property of Fischers protocol for the cases [17] $n = 2, \ldots, 8$. The time–performance of this experiment can be found in Table 10 and Figure 6. Execution times have been measured in seconds with the standard UNIX program `time`. We have also attempted to verify Fischers protocol using three other existing real–time verification tools: HyTech 0.6 [15], Kronos 1.1c [9], Epsilon 3.0 [6] using the same machine as for the UPPAAL experiment. As illustrated in Table 10 and Figure 6 the experiment showed that UPPAAL is significantly faster than all these tools (50–100 times) and able to deal with much larger systems; all the other tools failed [18] to verify Fischers protocol for more than 4 processes (indicated by $\perp$ in the Table).

The four tools can be devided into two categories: HyTech and Kronos both produce the product of the automata network before the verification is carried out, whereas Epsilon and UPPAAL verifies properties on–the–fly without ever explicitly producing the product automaton. A potential advantage of the first strategy is the reusability of the product automaton. The obvious advantage of the second strategy is that only the necessary part of product automaton needs to be examined saving not only time but also (more importantly) space. For HyTech and Kronos we have measured both the total time as well as the part spent on the actual verification (marked v in Table 10), i.e. not measuring the time for producing the product automaton.

[17] In fact we have verified the case of 9 processes, but on a different machine.

[18] Failure occured either because the verification ran out of memory, never terminated or did not accept the produced product automaton.

	2	3	4	5	6	7	8	9
HyTech	6.0	83.5	$\perp$					
HyTech^v	3.6	26.4	$\perp$					
Epsilon	0.8	10.6	242.6	$\perp$				
Kronos	0.5	4.0	50.5	$\perp$				
Kronos^v	0.2	3.4	46.9	$\perp$				
UPPAAL	0.2	0.2	0.7	5.5	18.8	145.0	1107.5	$\perp$

Table 10. Execution Times (seconds).

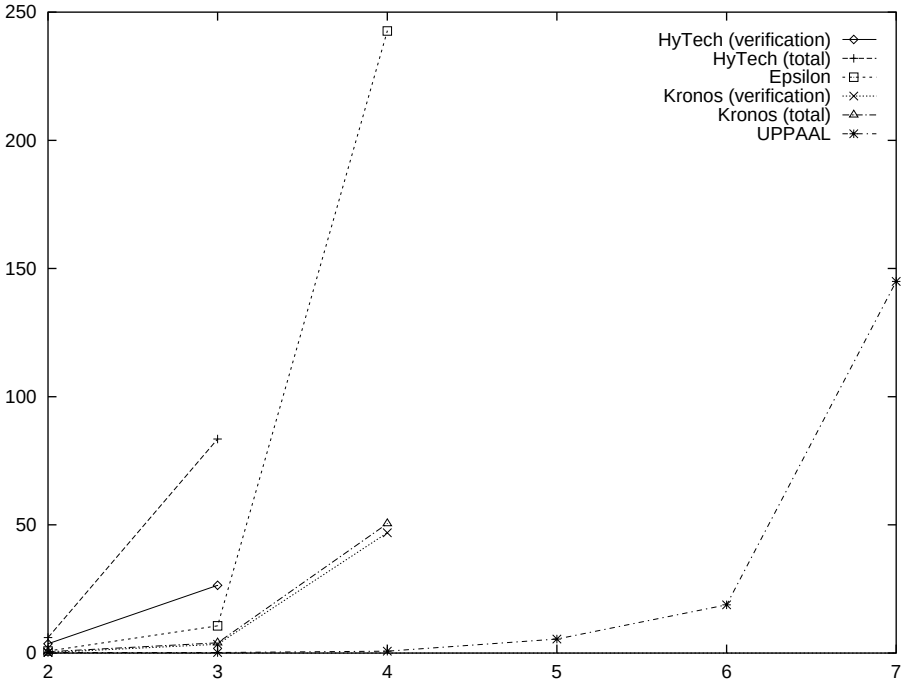


Fig. 6. Execution Times (seconds).

8 Conclusion and Future Work

In developing automatic verification algorithms for real-time systems, we need to deal with two potential types of explosion arising from parallel composition: explosion in the space of control nodes, and explosion in the region space over clock-variables. To attack these explosion problems, we have developed and combined compositional and symbolic model–checking techniques. These techniques have been implemented in a new automatic verification tool UPPAAL . Experimental results show that UPPAAL is not only substantially faster than other

real-time verification tools but also able to handle much larger systems.

We should point out that the safety logic we designed in this paper enables the presented techniques to be implemented in a very efficient way. Though the logic is less expressive than the full version of the timed μ–calculus T_μ, it is expressive enough to specify safety properties as well as bounded liveness properties. As future work, we shall study the practical applicability of this logic and UPPAAL by further examples. Our experience shows that the practical limits of UPPAAL is caused by the space–complexity rather than the time–complexity of the model–checking algorithms. Thus, future work includes development of more space–efficient methods for representation and manipulation of clock constraints. For a verification tool to be of practical use in a design process it is of out most importance that the tool offers some sort of diagnostic information in case of erroneous. Based on the synthesis technique presented in [12] we intend to extend UPPAAL with the ability to generate diagnostic information. Finally, more sophisticated minimization heuristics are sought to yield further improvement of our compositional technique.

Acknowledgment

The UPPAAL tool has been implemented in large parts by Johan Bengtsson and Fredrik Larsson. The authors would like to thank them for their excellent work. The first author would also like to thank Francois Laroussinie for several interesting discussions on the subject of compositional model–checking. The last two authors want to thank the Steering Committee members of NUTEK, Bengt Asker and Ulf Olsson, for useful feedback on practical issues.

References

1. Martin Abadi and Leslie Lamport. An Old–Fashioned Recipe for Real Time. *Lecture Notes in Computer Science*, 600, 1993.
2. R. Alur, C. Courcoubetis, and D. Dill. Model–checking for Real–Time Systems. In *Proceedings of Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.
3. R. Alur and D. Dill. Automata for Modelling Real–Time Systems. *Theoretical Computer Science*, 126(2):183–236, April 1994.
4. H. R. Andersen. Partial Model Checking. *To appear in Proceedings of LICS'95*, 1995.
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. K. Cerans, J. C. Godskesen, and K. G. Larsen. Timed modal specifications — theory and tools. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.
7. E. M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetry in Temporal Logic Model Checking. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.

8. E. M. Clarke, O. Grümberg, and D. E. Long. Model Checking and Abstraction. *Principles of Programming Languages*, 1992.
9. C. Daws, A. Olivero, and S. Yovine. Verifying ET–LOTOS programs with KRONOS. *In Proceedings of 7th International Conference on Formal Description Techniques*, 1994.
10. E. A. Emerson and C. S. Jutla. Symmetry and Model Checking. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.
11. P. Godefroid and P. Wolper. A Partial Approach to Model Checking. *Logic in Computer Science*, 1991.
12. J.C. Godskesen and K.G. Larsen. Synthesizing Distinghuishing Formulae for Real Time Systems — Extended Abstract. *Lecture Notes in Computer Science*, 1995. To occur in Proceedings of MFCS'95. Also BRICS report series RS–94–48.
13. Nicolas Halbwachs. Delay Analysis in Synchronous Programs. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.
14. T. A. Henzinger, Z. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Logic in Computer Science*, 1992.
15. Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The Cornell HYbrid TECHnology Tool. *To appear in the Proceedings of TACAS, Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1995.
16. H. Hüttel and K. G. Larsen. The use of static constructs in a modal process logic. *Lecture Notes in Computer Science, Springer Verlag*, 363, 1989.
17. F. Laroussinie, K. G. Larsen, and C. Weise. From Timed Automata to Logic — and Back. *Lecture Notes in Computer Science*, 1995. To occur in Proceedings of MFCS. Also BRICS report series RS–95–2.
18. F. Laroussinie and K.G. Larsen. Compositional Model Checking of Real Time Systems. *Lecture Notes in Computer Science*, 1995. To appear in Proceedings of CONCUR'95. Also BRICS report series RS–95–19.
19. N. Shankar. Verification of REal–Time Systems Using PVS. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.
20. A. Tarski. A lattice–theoretical fixpoint theorem and its applications. *Pacific Journal of Math.*, 5, 1955.
21. A. Valmari. A Stubborn Attack on State Explosion. *Theoretical Computer Science*, 3, 1990.
22. Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real–Time Systems By Constraint–Solving. *In the Proceedings of the 7th International Conference on Formal Description Techniques*, 1994.

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