Basic Research in Computer Science

BRICS RS-96-59 **Larsen et al.: Compositional and Symbolic Model-Checking of Real-Time Systems**

Compositional and Symbolic Model-Checking of Real-Time Systems

Kim G. Larsen
Paul Pettersson
Wang Yi

BRICS Report Series **RS-96-59**

ISSN 0909-0878 **December 1996**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through World Wide Web and anonymous FTP:

`http://www.brics.dk/`
`ftp://ftp.brics.dk/`
This document in subdirectory `RS/96/59/`

Compositional and Symbolic Model-Checking of Real-Time Systems*

Kim G. Larsen† Paul Pettersson‡ Wang Yi‡
Uppsala University

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.

In this paper we attack 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 indicate that* Uppaal *performs time- and space-wise favorably compared with other real-time verification tools.*

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 [1], by application of *partial order* methods [2, 3] which suppresses unnecessary interleavings of transitions, or by application of *abstractions* and *symmetries* [4, 5, 6].

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 [7] (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) [8] and timed μ-calculus (T_μ) [9]. 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 attack 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 description 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

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

†On leave from BRICS and the Department of Mathematics and Computer Science, Aalborg University, Fredrik Bajers Vej 7–E, DK–9220 Aalborg, Denmark. E-mail: kgl@iesd.auc.dk.

‡Department of Computer Systems, Box 325, Uppsala University, S751 05, Uppsala, Sweden. E-mail: {paupet,yi}@docs.uu.se.

to be verified and will thus in practice be considerably coarser (and smaller).

For the explosion on control-nodes, recent work by Andersen [10] on (untimed) finite-state systems gives experimental evidence that the quotient technique improves results obtained using Binary Decision Diagrams [1]. Our aim in this paper 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

$$\Big(A_1 \mid \ldots \mid A_n\Big) \models \varphi \quad \text{iff} \quad \Big(A_1 \mid \ldots \mid A_{n-1}\Big) \models \varphi \,/\, A_n \tag{1}$$

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

$$\Big(A_1 \mid \ldots \mid A_n\Big) \models \varphi \quad \text{iff} \quad \mathbf{1} \models \varphi \,/\, A_n \,/ A_{n-1} \,/\, \ldots \,/\, A_1 \tag{2}$$

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.

Our new symbolic and compositional verification technique is developed for a real-time logic designed specifically for expressing safety and bounded liveness properties. Comparatively less expressive than TCTL and T_μ, the logic is still sufficiently expressive for practical purposes, and the logic allows a number of operators of other logics to be derived. Most importantly, the somewhat restrictive expressive power of our logic allows for 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, [9] offers a symbolic verification technique. However, due to the high expressive power of these logics the partitioning employed in [9] is significantly finer (and larger) and implementation-wise more complicated than the symbolic technique we present in this paper. Our symbolic method is based on the constraint solving technique presented in [11], where the technique was developed for simple reachability problems.

An initial effort in applying the compositional quotienting technique to real-time systems has been given in [12]. This work also contains experimental evidence of the potential benefits of the quotient technique in a real-time setting. However, being based directly on the (very fine) notion of regions, [12] suffers from a potential explosion in the region-space.

The outline of this paper is as follows: In the next section we give a short presentation of the notions of timed automata and networks; in section 3, the safety logic is presented and its expressive power is illustrated. Section 4 describes the symbolic verification technique based on constraint solving and section 5 describes the compositional quotienting technique. Both techniques are illustrated by an example. In section 6, we report on our experimental results, which indicate that UPPAAL performs time- and space-wise favorably compared with other real-time verification tools.

2 Real-Time Systems

We shall use *timed transition systems* as a basic semantic 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* [11, 12].

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 $\mathcal{L}$ 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, $\longrightarrow \subseteq S \times \mathcal{L} \times S$ 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 [13] 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[1]. 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 [7] is a standard finite-state automaton extended with a finite collection of real-valued clocks[2]. 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.* □

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', n' are natural numbers. We use m, n, m', n' 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 $x \geq m$. 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

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

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

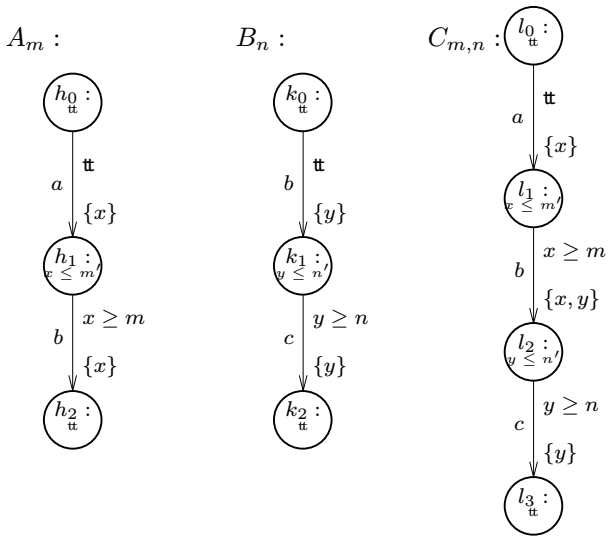

Figure 1: Three timed automata

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 of 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.

The semantic 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 $I(u')$

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 $d \ge 0$, $m \le e \le m'$ and $n \le f \le n'$. 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 $m \le e \le m'$ and $n \le f \le n'$ 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 isomorphic to the part of $A_m \otimes_f B_n$ which is reachable from (h_0, k_0). □

3 A Logic for Safety and Bounded Liveness Properties

It has been pointed out [14, 11], that the practical goal of verification of real-time systems, is to verify simple safety properties such as deadlock-freeness and mutual exlusion. Our previous work [11] shows that such properties can be verified on-the-fly by simple reachability analysis which avoids construction of the whole reachable state-space of systems.

3.1 Syntax and Semantics

We shall present a timed modal logic to specify safety properties. In fact, the logic can also be used to specify bounded liveness properties such as "whenever p becomes true, q will be true within a given time bound". The logic may be seen as a fragment of the timed μ-calculus presented in [9], and also studied in [15].

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 $\mathcal{L}_s$ of formulas over K, Id, Act, and $\mathcal{P}$ is generated by the abstract syntax with φ and ψ ranging over $\mathcal{L}_s$:*

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

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). □

As before, we shall use $\mathtt{tt}$ to stand for a formula like $x \geq 0$ which is always true, and $\mathtt{ff}$ for a formula $x < 0$ which is always false for a formula clock $x \in K$.

Note that the logic is essentially the fragment of the timed modal logic presented in [15] by eliminating existential quantification over delay transitions, general disjunction over formulas, and existential quantification over a-transitions.

We do allow a simple form of disjunction, in that a clock constraint or an atomic proposition may be disjuncted with an arbitrary formula. We disallow general disjunction in the logic to achieve efficient compositional and symbolic model-checking algorithms. However, the logic is expressive enough to specify safety and bounded liveness properties. We shall see, that the simple form of disjunction allows us to specify bounded liveness properties such as "p will be true within n".

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, we interpret the $\mathcal{L}_s$ 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 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 $\mathbb{V}\varphi$ means that all future states reachable from $\langle s, u \rangle$ by delays will satisfy property φ; $\mathbb{V}$ denotes universal quantification over delay transitions. Similarly, 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; $[a]$ denotes universal 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. Any relation satisfying the implications in Table 1 is called a *satisfiability* relation. It follows from standard fixpoint theory [16] that $\models_{\mathcal{D}}$ is the union of all satisfiability relations. For simplicity, we shall omit the index $\mathcal{D}$ and write $\models$ instead of $\models_{\mathcal{D}}$ whenever it is understood from the context.

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

Table 1: Definition of satisfiability.

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.

$$
\mathcal{F} = \left\{ \begin{array}{ll}
X_i \stackrel{\text{def}}{=} & [a]\Big(z \,\mathsf{in}\, Z_i\Big) \\
Z_i \stackrel{\text{def}}{=} & \mathsf{at}(l_3) \vee \\
 & \Big(z < i \wedge [a]Z_i \wedge [b]Z_i \wedge [c]Z_i \wedge \mathbb{V}Z_i\Big)
\end{array} \right\}
$$

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. It may be argued that $C_{m,n} \models X_{m'+n'}$ and (consequently), that $A_m \mid_f B_n \models X_{m'+n'}$. □

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 be a general formula, cp be an atomic clock constraint or an atomic proposition and n be a natural number. A collection of derived operators are given in Table 2.

$$
\begin{array}{rcl}
\mathsf{INV}(\varphi) & \equiv & X \text{ where } X \stackrel{\text{def}}{=} \varphi \wedge \Bbbv X \wedge [Act]X \\
\varphi \ \mathsf{UNTIL}\ cp & \equiv & X \text{ where} \\
 & & X \stackrel{\text{def}}{=} cp \vee \Big(\varphi \wedge \Bbbv X \wedge [Act]X\Big) \\
\varphi \ \mathsf{UNTIL}_{<n}\ cp & \equiv & z \ \mathsf{in}\ \Big((\varphi \wedge z < n)\ \mathsf{UNTIL}\ cp\Big) \\
cp \ \mathsf{BEFORE}\ n & \equiv & \mathsf{tt}\ \mathsf{UNTIL}_{<n}\ cp
\end{array}
$$

Table 2: Derived Operators

The intuitive meanings of these operators are as follows: $\mathsf{INV}(\varphi)$ is satisfied by a timed automaton means that the automaton must enjoy the property φ now, and for all future time points, the reachable states should satisfy $\mathsf{INV}(\varphi)$ (i.e. X), and after any action transition, the reachable states should again satisfy $\mathsf{INV}(\varphi)$ (i.e. X): namely that φ is an invariant property of the automaton. φ UNTIL cp is satisfied by a timed automaton means that the automaton enjoys the property cp now, or otherwise all reachable states by action transitions and delay transitions should satisfy φ. This simply means that φ must hold at least before cp becomes true. The bounded version of the UNTIL - construct φ $\mathsf{UNTIL}_{<n}$ cp is similar to φ UNTIL cp except that cp must be true within n time units. A simpler version of this operator is cp BEFORE n meaning that property cp must be true within n time units. Alternatively, φ $\mathsf{UNTIL}_{<n}$ cp can be defined as z in X where $X \stackrel{\text{def}}{=} cp \vee \Big(\varphi \wedge (x < n) \wedge \Bbbv X \wedge [Act]X\Big)$.

4 Symbolic Model-Checking

We have presented a model to describe real-time systems, i.e. networks of timed automata, and a logic to specify properties of such systems. The next question is how to check whether a given formula in the logic is satisfied by a given network of automata. This is the so-called model-checking problem. As the systems we are studying are in general infinite-state due to the real-valued clocks, we need efficient methods to represent the state-space symbolically. The region-graph technique by Alur, Courcoubetis and Dill 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. It follows that model-checking is decidable as the region partitioning enables standard finite-state algorithmic model-checking techniques to be applied. However, as the notion of region is property-independent and the number of such regions depends on the constants used in the clock constraints of an automaton, this leads to an extremely fine (and large) partitioning.

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 $\langle (l, u), v\rangle$ satisfies 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 $\mathcal{B}(C \cup K)$. Instead of checking $\langle (l, u), v\rangle \models \varphi$ for each u and v, 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.

4.1 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*

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

□

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 2. The three operations $A^{\uparrow}$, $A^{\downarrow}$ and $\{x\}A$ are illustrated in (b), (c) and (d) respectively of Figure 2. Intuitively, $A^{\downarrow}$ is the largest

set of time assignments that will eventually reach A after some delay; whereas $A^{\uparrow}$ is the dual of $A^{\downarrow}$: 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 $\{x_1...x_n\}(A) = \{x_1\}(\{x_2...x_n\}A)$.

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

Proposition 1 *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.

4.2 Model-Checking by Constraint Solving

Given a network of timed automaton A over clocks C, we shall interprete formulas 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 is defined as the largest relation satisfying the implications in Table 3. We call a relation satisfying the implications in Table 3 a *symbolic satisfiability* relation. Again, it follows from standard fixpoint theory [16] that $\vdash_{\mathcal{D}}$ is the union of all symbolic satisfiability relations. For simplicity, we shall omit the index $\mathcal{D}$ and write $\vdash$ instead of $\vdash_{\mathcal{D}}$ whenever it is understood from the context.

The following Theorem shows that the symbolic interpretation of $\mathcal{L}_s$ in Table 3 expresses the sufficient and necessary conditions for a timed automata to satisfy a formula φ[3].

[3]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.

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

Table 3: Definition of symbolic satisfiability.

Theorem 2 *Let A be a timed automaton over clock set C and φ a formula 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 $\{x = 0 \mid x \in C \cup K\}$.

Proof: It is proved by co-induction (on $\vdash$) that $[l, D] \vdash \varphi$ holds precisely when $\langle (l, u), v \rangle \models \varphi$ for all uv in D. □

Given a symbolic satisfaction *problem* $[l, D] \vdash \varphi$ we may determine its validity by using the implications of Table 3 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.

The operations and predicates on clock constraint systems discussed in Section 4.1 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 [17].

5 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

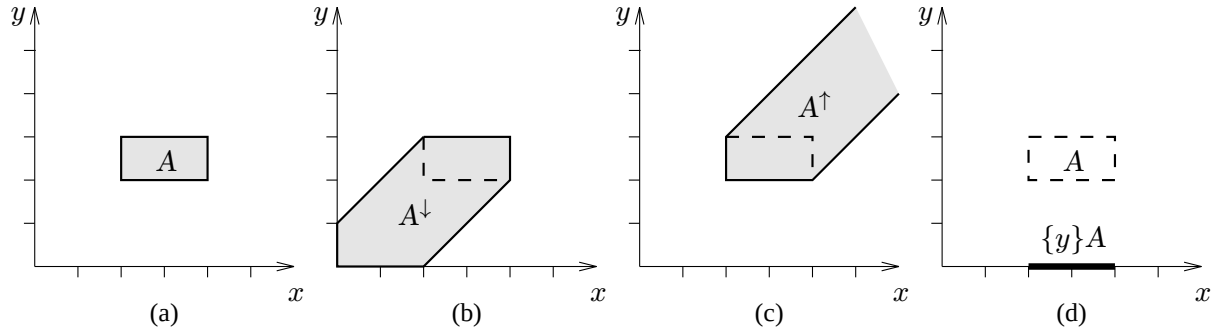

Figure 2: Operations on Solution Sets

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 [10] 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 [12] 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 [12] simple) quotient construction and show how to integrate it with the symbolic technique of the previous section.

5.1 Quotient Construction

Given a formula φ, and two timed automata A and B we aim at constructing a formula (called the *quotient*) $\varphi\big/_f B$ such that

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

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 $A \,|_f\, B$. 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 $\varphi\big/_f l$ over $C \cup K$ in Table 4 on the structure of φ[4][5].

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

Table 4: Definition of Quotient $\varphi\big/_f l$

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 $\mathbb{W}\varphi$ 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

[4] 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.

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

introduces new identifiers of the form X_l. These new identifiers and their definitions ($X_l \stackrel{\text{def}}{=} \mathcal{D}(X)\big/_f\, l$) are collected in the (quotient) declaration $\mathcal{D}_B$.

For l_0 the initial node of a timed automaton B, the quotient $\varphi\big/_f\, l_0$ expresses the sufficient and necessary requirement to a timed automaton A in order that the parallel composition $A\,|_f\, B$ satisfies φ. This is stated in the following Theorem 3:

Theorem 3 *Let A and B be two timed automata and let l_0 be the initial node of B. Then*

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

Example 5 Reconsider the network and synchronization function from Examples 1, 2 and 3. We want to establish that the network $A_m\,|_f\, B_n$ satisfies the following property Y provided $n + m \geq i$:

$$\begin{aligned} Y &\stackrel{\text{def}}{=} [a]\Big(z \;\mathsf{in}\; X\Big) \\ X &\stackrel{\text{def}}{=} (z \geq i) \vee \Big([c]\mathbf{ff} \wedge [a]X \wedge [b]X \wedge \mathbb{V}X\Big) \end{aligned}$$

The property Y 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 $n + m \geq i$.

From Theorem 3 it follows that the sufficient and necessary requirement to A_m in order that $A_m\,|_f\, B_n$ satisfies Y is that A_m satisfies $Y\big/_f\, k_0$. Using the quotient definition from Table 4 we get:

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

□

5.2 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

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

Table 5: Constraint Propagation

formulas and declarations. Below we describe the transformations considered:

Reachability: When considering an initial quotient formula $\varphi\big/_f\, l_0$ 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: $\mathbf{ff} \wedge \varphi \equiv \mathbf{ff}$, $\mathbf{tt} \wedge \varphi \equiv \varphi$, $\langle a\rangle\mathbf{ff} \equiv \mathbf{ff}$, $\exists\mathbf{ff} \equiv \mathbf{ff}$, $x \;\mathsf{in}\; \mathbf{ff} \equiv \mathbf{ff}$, $\langle a\rangle\varphi \wedge [a]\mathbf{ff} \equiv \mathbf{ff}$.

Constraint Propagation: Constraints on formula clocks may be propagated using various distribution laws (see Table 5). In some cases, propagation will lead to trivial clock constraints, which may be simplified to either $\mathbf{tt}$ or $\mathbf{ff}$ and hence made applicable to Boolean Simplification.

Constant Propagation: Identifiers with identifier-free definitions (i.e. constants such as $\mathbf{tt}$ or $\mathbf{ff}$) 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 $X = \mathbf{tt}$ 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 $\varphi[\mathbf{tt}/S]$[6] can be simplified to $\mathbf{tt}$. Then all identifiers of S can be removed provided the value $\mathbf{tt}$ is propagated to all uses of identifiers from S (as under Constant Propagation). The maximal set S

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

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[7]. 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 5. In particular, the strategies will find the quotient formula to be trivially true in certain cases.

Example 6 Reconsider Example 5 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 5 we get:

$$\begin{aligned} \mathtt{tt} \Rightarrow \Big(y \,\mathsf{in}\, Y_0\Big) &\equiv \mathtt{tt} \Rightarrow \Big(\{y,z\} \,\mathsf{in}\, X_0\Big) \\ &\equiv \{y,z\} \,\mathsf{in}\, \Big(D_0 \Rightarrow X_0\Big) \end{aligned}$$

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

$$\begin{aligned} &(D_0 \Rightarrow X_0) \\ &\quad\equiv D_0 \Rightarrow \Big[(z \geq i) \vee \Big([b](y \,\mathsf{in}\, X_1) \wedge \mathbb{V} X_0\Big)\Big] \\ &\quad\equiv \Big(D_0 \Rightarrow [b](y \,\mathsf{in}\, X_1)\Big) \wedge \Big(D_0 \Rightarrow \mathbb{V} X_0\Big)^{8} \\ &\quad\equiv [b]\Big(y \,\mathsf{in}\, (D_0 \Rightarrow X_1)\Big) \wedge \mathbb{V}\Big({D_0}^{\uparrow} \Rightarrow X_0\Big) \end{aligned}$$

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

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

Table 6: Equations after Constraint Propagation

Continuing constraint propagation yields the equations in Table 6, where $D_1 = (y = 0 \wedge z < i)$. □

Example 7 (Example 6 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 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 6 we get equations which after application of Boolean Simplification and Trivial Equation Elimination all simplifies 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 $A \,|_f B_n$ satisfies Y. □

6 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

[8] Note that $(z < i \wedge D_0) = D_0$.

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 faster than the other tools but also able to handle larger systems.

In particular, we have used the so called Fischer's mutual exclusion protocol [17, 18, 19] 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.

6.1 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 Fischer's protocol for the cases[9] $n = 2, \ldots, 8$. The time-performance of this experiment can be found in Figure 3. Execution times have been measured in seconds with the standard UNIX program `time`. We have also attempted to verify Fischer's protocol using three other existing real-time verification tools: HyTech [20] (version 0.6 and version 1.0), Kronos 1.1c [21] and Epsilon 3.0 [22] using the same machine as for the UPPAAL experiment. As illustrated in Figure 3 the experiment showed that UPPAAL is faster than all these tools and able to deal with larger systems; all the other tools failed[10] to verify Fischer's protocol for more than 5 processes.

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 (recently another on-the-fly verification technique for timed automata has been studied in [23]). 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

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

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

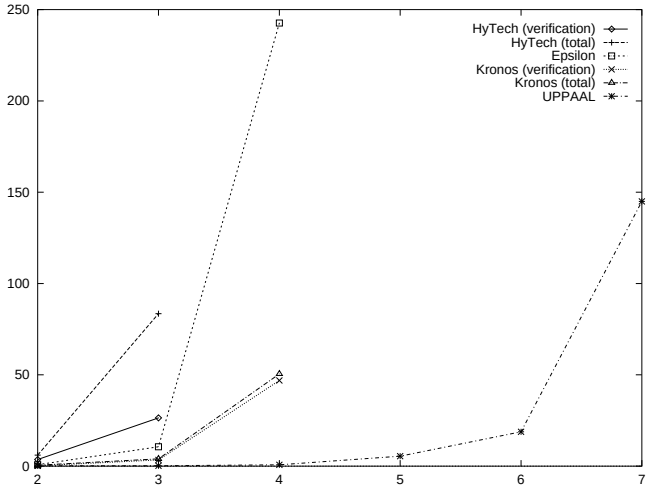

Figure 3: Execution Times.

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 i.e. not measuring the time for producing the product automaton.

7 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 faster than other real-time verification tools but also able to handle 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 most importance that the tool offers some sort of diagnostic information in case of errors. Based on the synthesis technique presented in [24] 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] 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.

[2] P. Godefroid and P. Wolper. A Partial Approach to Model Checking. *Logic in Computer Science*, 1991.

[3] A. Valmari. A Stubborn Attack on State Explosion. *Theoretical Computer Science*, 3, 1990.

[4] 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.

[5] E. M. Clarke, O. Grümberg, and D. E. Long. Model Checking and Abstraction. *Principles of Programming Languages*, 1992.

[6] E. A. Emerson and C. S. Jutla. Symmetry and Model Checking. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.

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

[8] 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.

[9] T. A. Henzinger, Z. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Logic in Computer Science*, 1992.

[10] H. R. Andersen. Partial Model Checking. *In Proc. of LICS'95*, 1995.

[11] 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.

[12] F. Laroussinie and K.G. Larsen. Compositional Model Checking of Real Time Systems. *Lecture Notes in Computer Science*, 1995. Proc. of CONCUR'95.

[13] 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.

[14] Nicolas Halbwachs. Delay Analysis in Synchronous Programs. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.

[15] F. Laroussinie and K.G. Larsen. From Timed Automata to Logic — and Back. *Lecture Notes in Computer Sciencie*, 1995. Proc. of MFCS'95.

[16] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Math.*, 5, 1955.

[17] Kim G. Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In *Proc. of Fundamentals of Computation Theory*, 1995.

[18] Martin Abadi and Leslie Lamport. An Old-Fashioned Recipe for Real Time. *Lecture Notes in Computer Science*, 600, 1993.

[19] N. Shankar. Verification of Real-Time Systems Using PVS. *Lecture Notes in Computer Science*, 697, 1993. In Proceedings of CAV'93.

[20] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The Cornell HYbrid TECHnology Tool. *Proc. of TACAS, Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1995. BRICS report series NS–95–2.

[21] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. *In Proceedings of 7th International Conference on Formal Description Techniques*, 1994.

[22] 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.

[23] Oleg V. Sokolsky and Scott A. Smolka. Local model checking for real-time systems. In *Proc. of CAV'95*, volume 939, pages 211–224. Springer Verlag, 1995.

[24] J.C. Godskesen and K.G. Larsen. Synthesizing Distinghuishing Formulae for Real Time Systems — Extended Abstract. *Lecture Notes in Computer Science*, 1995. Proc. of MFCS'95.

Recent Publications in the BRICS Report Series

RS-96-59 Kim G. Larsen, Paul Pettersson, and Wang Yi. *Compositional and Symbolic Model-Checking of Real-Time Systems*. December 1996. 12 pp. Appears in *16th IEEE Real-Time Systems Symposium*, RTSS '95 Proceedings, 1995.

RS-96-58 Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — *a Tool Suite for Automatic Verification of Real–Time Systems*. December 1996. 12 pp. Appears in Alur, Henzinger and Sontag, editors, *DIMACS Workshop on Verification and Control of Hybrid Systems*, HYBRID '96 Proceedings, LNCS 1066, 1996, pages 232–243.

RS-96-57 Kim G. Larsen, Paul Pettersson, and Wang Yi. *Diagnostic Model-Checking for Real-Time Systems*. December 1996. 12 pp. Appears in Alur, Henzinger and Sontag, editors, *DIMACS Workshop on Verification and Control of Hybrid Systems*, HYBRID '96 Proceedings, LNCS 1066, 1996, pages 575–586.

RS-96-56 Zine-El-Abidine Benaissa, Pierre Lescanne, and Kristoffer H. Rose. *Modeling Sharing and Recursion for Weak Reduction Strategies using Explicit Substitution*. December 1996. 35 pp. Appears in Kuchen and Swierstra, editors, *8th International Symposium on Programming Languages, Implementations, Logics, and Programs*, PLILP '96 Proceedings, LNCS 1140, 1996, pages 393–407.

RS-96-55 Kåre J. Kristoffersen, François Laroussinie, Kim G. Larsen, Paul Pettersson, and Wang Yi. *A Compositional Proof of a Real-Time Mutual Exclusion Protocol*. December 1996. 14 pp. To appear in Dauchet and Bidoit, editors, *Theory and Practice of Software Development. 7th International Joint Conference CAAP/FASE*, TAPSOFT '97 Proceedings, LNCS, 1997.

RS-96-54 Igor Walukiewicz. *Pushdown Processes: Games and Model Checking*. December 1996. 31 pp. Appears in Alur and Henzinger, editors, *8th International Conference on Computer-Aided Verification*, CAV '96 Proceedings, LNCS 1102, 1996, pages 62–74.