CMC: A TOOL FOR COMPOSITIONAL MODEL-CHECKING OF REAL-TIME SYSTEMS

François Laroussinie[1]
and Kim G. Larsen[2]

[1]Laboratoire Spécification et Vérification,
ENS de Cachan, CNRS URA 2236
61, av. Pdt Wilson, 94235 Cachan Cedex, France
fl@lsv.ens-cachan.fr

[2]Basic Research in Computer Science
Centre of the Danish National Research Foundation.
Aalborg University, Fredrik Bajers Vej 7-E, DK-9220 Aalborg, Denmark
kgl@cs.auc.dk

Abstract: In this paper we present a tool (CMC) for compositional model-checking of real-time systems. CMC is based on a completely different method compared to existing real-time verification tools (HYTECH, KRONOS, UPPAAL). After a description of the method, we illustrate its efficiency by considering two examples : the Fischer's mutual exclusion protocol and a railroad crossing system.

INTRODUCTION

Within the last decade, model-checking has turned out to be a useful and successful technique for the verification of temporal properties in finite state systems. More recently, serious attempts have been made to extend the success of model-checking to the setting of real-time systems, with timed automata [2] as models. The major obstacle for the model-checking approach is the well-known state explosion problem due to parallel composition (as in the untimed case) and also to time encoding. Several heuristics have been proposed to overcome this problem : symbolic model-checking [10], on-the-fly technique [6, 19], efficient data-structures for time constraints [17, 4]. These and other techniques have been implemented into tools (KRONOS [22], UPPAAL [19] and HYTECH [11]) and a number of industrial case studies have been analyzed and verified using these tools [12, 13, 5, 20, 9].

In [16] we proposed a first *compositional model-checking* technique, which in a number of practical examples seems to avoid the state explosion problem. In this method, we never construct nor examine the global (symbolic) state space of a real-time system modeled as a network $(A_1|\ldots|A_n)$ of timed automata. Rather, when checking the system for a property φ, the components $A_1, \ldots, A_n$ are gradually removed from the network and incorporated into the formula; thus, the component A_n could be initially transfered from the network into the property φ yielding a new *quotient* property φ/A_n which expresses the precise property which has to be satisfied by the remaining network $(A_1|\ldots|A_{n-1})$ in order that the original property φ holds for the complete system. Now by repeatedly quotienting components from the network into the formula, we will finally be faced with the problem of establishing that the process *nil* (a process capable of performing no action at all) satisfies the last formula. A key point of this approach is to develop efficient simplification strategies in order to keep a small size of the quotiented formula.

In this paper we extend the compositional method in several ways. Firstly, we develop the method to a larger class of timed automata (with invariants and atomic propositions). Secondly we introduce a completely new (and considerably more efficient) quotient construction and specific simplification techniques adapted to our formalisms. Finally we present CMC a new real-time verification tool implementing the compositional method.

Plan of the paper: we first introduce the formalisms used in CMC : networks of timed automata used for the modeling of real-time systems, and the timed modal logic L_ν for specifying properties. We then present the quotient construction together with the simplification strategies emphasizing those dealing with clock constraints. Finally we illustrate the method with two examples. The first example is Fischer's mutual exclusion protocol [1, 21], which is a classical benchmark in this domain. Here our results both demonstrate the efficiency as well as the limits of the compositional approach. The second example is the railroad crossing system. In this example, L_ν proves to be an extremely convenient specification language, and our results proves that compositional method is useful for arbitrary systems and not only for networks with a symmetrical structure as in Fischer's protocol.

NETWORKS OF TIMED AUTOMATA

Let $\mathcal{A}$ be a finite set of actions and AP a finite set of atomic propositions. We denote by $\mathbf{N}$ the set of natural numbers and by $\mathbf{R}$ the set of non–negative real numbers. Θ denotes the set of delay actions $\{\epsilon(d) \mid d \in \mathbf{R}\}$. If C is a set of clocks, $\mathcal{B}(C)$ denotes *clocks constraints* over C, that is the set of formulas built using boolean connectives over atomic formulas of the form $x \bowtie m$ or $x - y \bowtie m$ with $x, y \in C$, $m \in \mathbf{N}$ and $\bowtie \in \{=, <, >, \leq, \geq\}$. Moreover $\mathcal{B}^-(C)$ denotes the

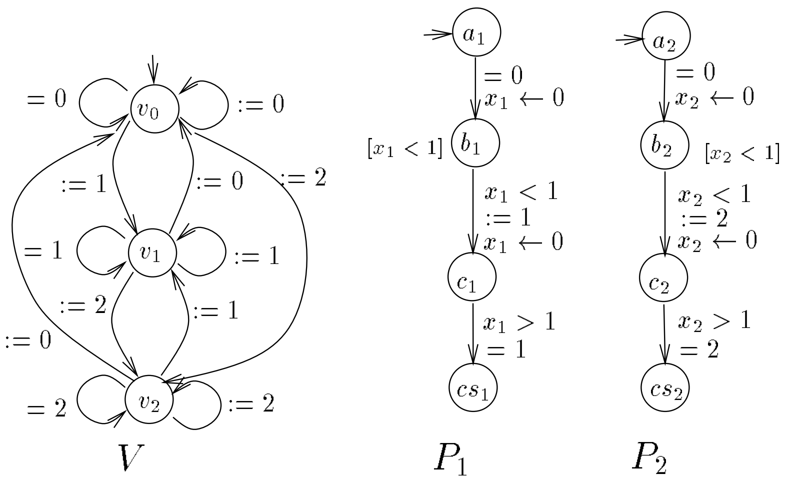


Figure 1.1 The Fischer's mutual exclusion protocol modeled with timed automata

restriction of $\mathcal{B}(C)$ to the boolean value *true* and to the conjunctions of atomic formulas of the form $x \bowtie m$ with $x \in C$ and $\bowtie \in \{<, \leq\}$. A time assignment v for C is a function from C to $\mathbf{R}$. We denote by $\mathbf{R}^C$ the set of time assignments for C. For $v \in \mathbf{R}^C$ and $d \in \mathbf{R}$, $v + d$ denotes the time assignment which maps each clock x in C to the value $v(x) + d$. For $C' \subseteq C$, $[C' \mapsto 0]v$ denotes the assignment for C which maps each clock in C' to the value 0 and agrees with v over $C \backslash C'$. Whenever $v \in \mathbf{R}^C$, $u \in \mathbf{R}^K$ and C and K are disjoint, vu denotes the time assignment over $C \cup K$. Given a condition $g \in \mathcal{B}(C)$ and a time assignment $v \in \mathbf{R}^C$, we note $v \models g$ when g holds for v. We define the timed automata (TA):

Definition 1 *A timed automaton A over $\mathcal{A}$ is a tuple $\langle N, \eta_0, C, E, l, Inv \rangle$ where N is a finite set of nodes, η_0 is the initial node, C is a finite set of clocks, $E \subseteq N \times \mathcal{B}(C) \times \mathcal{A} \times 2^C \times N$ corresponds to the set of edges: $e = \langle \eta, g, a, r, \eta' \rangle \in E$ represents an edge from the node η to the node η' with action a, r denotes the set of clocks to be reset and g is the enabling condition (the guard) over the clocks of A. We use the notation $\eta \xrightarrow{g,a,r} \eta'$. Moreover $l : N \mapsto 2^{\mathsf{AP}}$ is a labeling function of nodes with atomic propositions. Finally $Inv : N \mapsto \mathcal{B}^-(C)$ is a function assigning an invariant to each node.*

Example 1 *Consider the automata V, P_1 and P_2 in Figure 1.1. The automaton P_1 has four nodes, a_1, b_1, c_1 and cs_1, one clock x_1 and three edges. The edge between b_1 and c_1 has ":= 1" as action, $\{x_1\}$ as reset set and the enabling condition for the edge is $x_1 < 1$. Node b_1 has the invariant $x_1 < 1$ to enforce progress from the node before one time-unit elapses. The default invariant (for nodes a_1, c_1 and cs_1) is true.*

Note that the automaton V is a "classical" automaton (without clock). □

A *state* or a *configuration* of an automaton A is a pair (η, v) where η is a node of A and v a time assignment for C. Informally, the system starts at node η_0 with all its clocks initialized to 0 (i.e. with the time assignment v_0). The values of the clocks may increase synchronously with time as long as the invariant requirement $Inv(\eta_0)$ holds for the current values of the clocks . At any time, the automaton whose current node is η can change node by following an edge $\langle \eta, g, a, r, \eta' \rangle \in E$ provided the current values of the clocks satisfy g. With this transition the clocks in r get reset to 0. Formally the semantics of A is defined as a *labeled timed transition system*:

Definition 2 *A labeled timed transition system over $\mathcal{A}$ is a tuple $\mathcal{S} = \langle S, s_0, \rightarrow, l \rangle$, where S is a set of states, s_0 is the initial state, $\rightarrow \subseteq S \times \mathcal{A} \cup \Theta \times S$ is a transition relation and $l : S \mapsto 2^{\mathsf{AP}}$ a labeling function. We require that for any $s \in S$ and $d \in \mathbf{R}$, there exists at most one state s^d such that $s \xrightarrow{\epsilon(d)} s^d$. Moreover if s^d exists, then $s^{d'}$ exists for any $0 \leq d' \leq d$. Finally $(s^d)^e = s^{d+e}$.*

The labeled timed transition system associated with A is $\mathcal{S}_A = \langle S_A, \sigma_0, \longrightarrow_A \rangle$, where S_A is the set of states of A, σ_0 is the initial state (η_0, v_0), and $\longrightarrow_A$ is the transition relation defined as follows:

$$
\begin{array}{lll}
(\eta, v) \xrightarrow{a} (\eta', v') & \text{iff} & \exists \, \langle \eta, g, a, r, \eta' \rangle \in E \ \text{ s.t. } \ v \models g \ , \ v' \models Inv(\eta') \ \text{ and} \\
 & & v' = [r \mapsto 0]v \\
(\eta, v) \xrightarrow{\epsilon(d)} (\eta', v') & \text{iff} & \eta = \eta' \ \text{ and } \ v' = v + d \ \text{ and } (v + d) \models Inv(\eta)
\end{array}
$$

The labeling with atomic propositions is defined by: $l(\langle \eta, v \rangle) = l(\eta)$. Note that the condition $(v + d) \models Inv(\eta)$ in the second rule implies that for any d' s.t. $0 \leq d' \leq d$ we have $(v + d') \models Inv(\eta)$ because invariants are conjunctions of $x < k$ or $x \leq k$ formula.

Real-time systems may be modeled as parallel composition of timed automata. Following [14] we suggest a composition parameterized with a synchronization function generalizing a large range of existing notions of parallel compositions. Let $A_1, \ldots, A_n$ be n timed automata. A *synchronization function* f is a partial function $(\mathcal{A} \cup \{\bullet\}) \times \ldots \times (\mathcal{A} \cup \{\bullet\}) \hookrightarrow \mathcal{A}$, where $\bullet$ denotes a distinguished no–action symbol (for any i and $\eta \in N_i$, we assume $(\eta, \mathtt{tt}, \bullet, \emptyset, \eta) \in E_i$). In fact, f is an n-ary synchronization function with renaming. We denote by $(A_1 | \ldots | A_n)_f$ the parallel composition of $A_1, \ldots, A_n$ w.r.t. the synchronization function f. A network configuration is a pair $\langle \bar{\eta}, v \rangle$ where $\bar{\eta} = (\eta_1, \ldots, \eta_n)$ is vector of nodes and v is a valuation for $C = \cup_i C_i$, i.e. the clocks of the network (we denote by v_i the restriction $v_{|C_i}$).
The semantics of $(A_1 | \ldots | A_n)_f$ can be defined as a labeled timed transition system whose states are the configurations of the network and the transitions are given by the two following rules:

$$
\begin{array}{lll}
\langle \bar{\eta}, v \rangle \xrightarrow{\epsilon(d)} \langle \bar{\eta}, v + d \rangle & \text{iff} & \forall i \in \{1, \ldots, n\}, \langle \eta_i, v_i \rangle \xrightarrow{\epsilon(d)} \langle \eta_i, v_i + d \rangle \\
\langle \bar{\eta}, v \rangle \xrightarrow{b} \langle \bar{\eta'}, v' \rangle & \text{iff} & \forall i \in \{1, \ldots, n\}, \langle \eta_i, v_i \rangle \xrightarrow{a_i} \langle \eta'_i, v'_i \rangle \text{ s.t. } a_i \in \mathcal{A} \cup \{\bullet\} \\
 & & \text{and } v' = v'_1 \ldots v'_n \text{ and } f(a_1, \ldots, a_n) = b
\end{array}
$$

Note that the first rule stipulates that all clocks increase synchronously. The atomic proposition labeling is done in a natural way: $l(\langle \bar{\eta}, v \rangle) = \cup_i l(\eta_i)$.

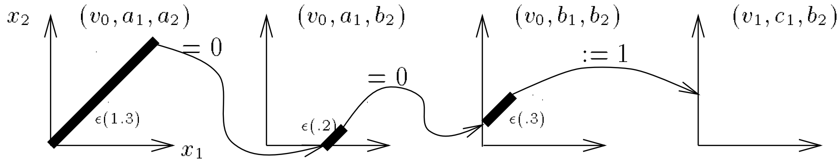


Figure 1.2 A prefix of an execution for FP.

Example 2 *Consider again the automata V, P_1 and P_2 of figure 1.1. This system describes Fischer's mutual exclusion protocol: two processes (modeled by P_1 and P_2) both try to reach a critical section (cs_1 and cs_2 resp.). To guarantee mutual exclusion of the critical sections, P_1 and P_2 share a variable (modeled by the process V) which can be assigned to 0,1 or 2 by the processes. Moreover the current value of V can be read by the two processes.*
The protocol can now be modeled by the parallel composition $FP = (V|P_1|P_2)_f$ with f the ternary synchronization function defined as follow: $f(=0,=0,\bullet) \stackrel{def}{=} f(=0,\bullet,=0) \stackrel{def}{=} =0$, $f(a,a,\bullet) \stackrel{def}{=} a$ for any a in $\{=1, :=1\}$ and $f(a,\bullet,a) \stackrel{def}{=} a$ for any a in $\{=2, :=2\}$. The values of f correspond to the name of the visible action.
The coordinate systems in Figure 1.2 indicates (some of) the states of the system FP. Each point of the coordinate systems represents a unique time assignment for x_1 and x_2, with the coordinate systems representing states involving the nodes configurations (v_0, a_1, a_2), (v_0, a_1, b_2), (v_0, b_1, b_2) and $(v1, c1, b2)$ respectively. We indicated the following transition sequence : $((v_0,a_1,a_2),(0,0)) \xrightarrow{\epsilon(1.3)} ((v_0,a_1,a_2),(1.3,1.3)) \xrightarrow{=0} ((v_0,a_1,b_2),(1.3,0)) \xrightarrow{\epsilon(0.2)} ((v_0,a_1,b_2),(1.5,0.2)) \xrightarrow{=0} ((v_0,b_1,b_2),(0,0.2)) \xrightarrow{\epsilon(0.3)} \ldots$ □

SPECIFYING TIMED SYSTEMS IN CMC

In our tool CMC, specifications of real-time systems are given as properties of the timed modal logic L_ν [16]. This logic may be seen as a fragment of the logic T_μ of [10] allowing only maximal fixed-points. Even though this restriction only enables safety properties to be specified and rules out liveness properties in general, it is still expressive enough to specify time-bounded liveness properties. In many real-time applications this is precisely what we wanted. Avoiding alternations of fixed-points also simplifies our model-checking algorithm.

Definition 3 *Let K a finite set of clocks, Id a set of identifiers, AP a set of atomic propositions, and $\mathcal{A}$ a set of actions. The set L_ν of formulae over K, Id, AP and $\mathcal{A}$ is generated by the following grammar:*

$$\begin{array}{rcl} L_\nu \ni \psi, \varphi & ::= & \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle a \rangle \varphi \mid [a] \varphi \mid \langle \delta \rangle \varphi \mid [\delta] \varphi \mid Z \\ & & \mid x \ \mathsf{in} \ \varphi \mid x + m \bowtie y + m' \mid x \bowtie m \mid ap \mid \neg ap \end{array}$$

where $a \in \mathcal{A}$; $x, y \in K$; $m, m' \in \mathbf{N}$; $\bowtie \in \{=, <, \leq, >, \geq\}$, $Z \in \mathsf{Id}$ and $ap \in \mathsf{AP}$.

$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} \varphi \wedge \psi$	iff	$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} \varphi$ and $\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} \psi$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} \langle a\rangle\, \varphi$	iff	$\exists\, \langle \bar{\eta'}, v', u\rangle.\ \langle \bar{\eta}, v\rangle \xrightarrow{a} \langle \bar{\eta'}, v'\rangle$ and $\langle \bar{\eta'}, v', u\rangle \models_{\mathcal{D}} \varphi$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} \langle \delta\rangle\ \varphi$	iff	$\exists d \in \mathbf{R}$ s.t. $\langle \bar{\eta}, v\rangle \xrightarrow{\epsilon(d)} \langle \bar{\eta}, v+d\rangle$ and $\langle \bar{\eta}, v+d, u+d\rangle \models_{\mathcal{D}} \varphi$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} [\delta]\ \varphi$	iff	$\forall d \in \mathbf{R}$, $\langle \bar{\eta}, v\rangle \xrightarrow{\epsilon(d)} \langle \bar{\eta}, v+d\rangle$ implies $\langle \bar{\eta}, v+d, u+d\rangle \models_{\mathcal{D}} \varphi$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} x \bowtie m$	iff	$u(x) \bowtie m$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} x \,\mathsf{in}\, \varphi$	iff	$\langle \bar{\eta}, v, [\{x\} \mapsto 0]u\rangle \models_{\mathcal{D}} \varphi$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} Z$	iff	$\langle \bar{\eta}, v, u\rangle$ belongs to the maximal solution of $Z = \mathcal{D}(Z)$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} p$	iff	$\exists i$ s.t. $p \in l(\eta_i)$
$\langle \bar{\eta}, v, u\rangle \models_{\mathcal{D}} A_j : p$	iff	$p \in l(\eta_j)$

Table 1.1 Semantics of L_ν.

Note that we can easily define new operators as $\mathtt{tt}$, $\mathtt{ff}$, $\varphi \Rightarrow \psi$ (if no identifier occur in φ). The meaning of the identifiers is specified by a declaration $\mathcal{D}$ assigning a formula of L_ν to each identifier. Given $S = (A_1 | \ldots | A_n)_f$ a network of TA, we interpret the formulas of L_ν with respect to *extended configurations* $\langle \bar{\eta}, v, u\rangle$, where $\langle \bar{\eta}, v\rangle$ is a configuration of S and u is a time assignment for K. Whereas the classical modal operators $\langle a\rangle$ and $[a]$ deal with *action* transitions, the operator $\langle \delta\rangle$ (resp. $[\delta]$) denotes existential (resp. universal) quantification over *delay* transitions.
The K clocks are so-called formula clocks, used as stopwatches for measuring the time elapsing between states (specified as properties) of the system. The formula clocks increase synchronously with the automata clocks. The formula $(x \,\mathsf{in}\, \varphi)$ initializes the formula clock x to 0 and the formula $(x \bowtie m)$ are used to compare the value of x in the current extended configuration with an integer value.
Finally, an extended configuration satisfies an identifier Z if it belongs to the maximal solution of the equation $Z = \mathcal{D}(Z)$. The atomic propositions can be of the form p or $A_j : p$ (to specify that p holds for component A_j) and they can be in the scope of a negation. The formal definition of L_ν semantics is given in table 1.1 ($\vee$ and $[a]$ are dual cases of $\wedge$ and $\langle a\rangle$).
Within L_ν we may express a number of classical temporal (and timed) properties. Let $\mathcal{B}$ be a subset of $\mathcal{A}$. We have the following derived operators :

- $\mathsf{ALWAYS}_{\mathcal{B}}(\varphi)$ expresses that φ holds from any state reachable using actions of $\mathcal{B}$ and delay transitions. The AG operator of CTL corresponds to $\mathsf{ALWAYS}_{\mathcal{A}}$. $\mathsf{ALWAYS}_{\mathcal{B}}(\varphi)$ can be defined by the following equation: $X \stackrel{\text{def}}{=} \varphi \wedge \bigwedge_{a \in \mathcal{B}} [a]X \wedge [\delta]\ X$.
- $\varphi_1 \mathsf{Until}_{\mathcal{B}} \varphi_2$ corresponds to the weak until operator of temporal logic ("φ_1 holds unless φ_2 holds, when following delay transitions and actions within

$\mathcal{B}$"). To define $\mathsf{Until}_{\mathcal{B}}$, we can use the equation:
$X \stackrel{\text{def}}{=} \varphi_2 \vee \Big(\varphi_1 \wedge [\delta]\ X \wedge \bigwedge_{a \in \mathcal{B}} [a]X\Big).$

- $\varphi \mathsf{UpTo}\ d$ ("φ will hold during at least d time units, whatever the performed transitions") can be expressed with: $x\ \mathsf{in}\ (\varphi \mathsf{Until}_{\mathcal{A}}(x > d))$.

Example 3 *Let* $\mathsf{Mut}_{(1,2)}$ *be the specification stating that* P_1 *and* P_2 *can never reach the critical section at the same time (see example 2). To write such a specification we need to label the* cs_i *nodes by the atomic proposition* CS*. We have:* $\mathsf{Mut}_{(1,2)} = \mathsf{ALWAYS}_{\mathcal{A}}(\neg P_1 : CS \vee \neg P_2 : CS)$ *with* $\mathcal{A} = \{=0, =1, =2, :=1, :=2\}$. □

COMPOSITIONAL MODEL-CHECKING

Quotient Principles

Recall from the Introduction that we are interested in constructing φ/A_n such that $(A_1|\ldots|A_n) \models \varphi$ if and only if $(A_1|\ldots|A_{n-1}) \models \varphi/A_n$. A first quotient construction for L_ν (dealing with timed automata without invariant nor atomic proposition) has been defined in [16] and a more efficient quotienting for a sublogic L_s has been given in [18, 15]. Here we present this new quotienting extended to L_ν.

A network of timed automata is described by using a n-ary synchronization function but it is easier to consider binary synchronization between A_n and the remaining part to define the quotient construction. Since our synchronization function allows renaming, we can easily decompose a parallel composition $(A_1|\ldots|A_n)_f$ as: $((A_1|\ldots|A_{n-1})_{f''}|A_n)_{f'}$ where f' (resp. f'') is a binary (resp. $(n-1)$-ary) synchronization function.

In table 1.2, we define the quotient construction (the cases $[a]\varphi$ and $\varphi \vee \psi$ are dual cases of those presented), with the following convention: $\langle \bullet \rangle \varphi = [\bullet]\varphi = \varphi$. Moreover the formula $r\ \mathsf{in}\ \varphi$ with $r = \{c_1, \ldots, c_l\} \subseteq C$ stands for $c_1\ \mathsf{in}\ \ldots c_l\ \mathsf{in}\ \varphi$. We have the following theorem [1]:

Theorem 1 *Let* $A_1, \ldots, A_n$ *be* n *timed automata* $\langle N_i, \eta_{i,0}, C_i, E_i, l_i, Inv_i \rangle$*. Let* φ *be a* L_ν *formula over the set of clocks* K*, all over an action set* $\mathcal{A}$*. Let* $\langle(\bar\rho, \eta), vw, u\rangle$ *be an extended configuration of* $((A_1|\ldots|A_{n-1})_{f'}|A_n)_f$ *with* $v \in \mathbf{R}^{\cup_{i<n} C_i}$*,* $w \in \mathbf{R}^{C_n}$ *and* $u \in \mathbf{R}^K$*, then we have:*

$$\Big\langle (\bar\rho, \eta),\ v\,w,\ u \Big\rangle \models_{\mathcal{D}} \varphi \quad \textit{if and only if} \quad \Big\langle \bar\rho,\ v,\ w\,u \Big\rangle \models_{\mathcal{D}'} \varphi\big/_{f}\ \eta$$

Note that the clocks in C_n become formula clocks after the quotienting. Moreover the quotienting rule for variables may increase the number of equations (in the declaration $\mathcal{D}'$ compared with the declaration $\mathcal{D}$) since it creates a new variable Z^η for any variable Z in φ and any node η of the quotiented automaton. In order to avoid trading the acknowledged explosion problem of the global state space with a similarly harmfull explosion problem in the size of quotient formulas, it is essential that we find strategies for simplifying formulas. Note

$(\varphi_1 \wedge \varphi_2)/_f\, \eta = (\varphi_1/_f\, \eta) \wedge (\varphi_2/_f\, \eta)$	$Z/_f\, \eta = Z^\eta$
$\Big(\langle a\rangle\varphi\Big)/_f\, \eta = \bigvee_{\eta \xrightarrow{g,c,r} \eta' \in E_n\, s.t.\, f(b,c)=a} g \wedge (r \text{ in } Inv(\eta')) \wedge \langle b\rangle \Big(r \text{ in } \varphi/_f\, \eta'\Big)$	
$\langle\delta\rangle\, \varphi/_f\, \eta = \langle\delta\rangle \Big(Inv(\eta) \wedge \varphi/_f\, \eta\Big)$	$(x \bowtie c)/_f\, \eta = (x \bowtie c)$
$[\delta]\ \varphi/_f\, \eta = [\delta]\ \Big(Inv(\eta) \Rightarrow \varphi/_f\, \eta\Big)$	$(x \text{ in } \varphi)/_f\, \eta = x \text{ in } \Big(\varphi/_f\, \eta\Big)$
$(P:\alpha)/_f\, \eta = \begin{cases} \mathsf{tt} & \text{iff } P = A_n \ \wedge\ \alpha \in l_{A_n}(\eta) \\ \mathsf{ff} & \text{iff } P = A_n \ \wedge\ \alpha \notin l_{A_n}(\eta) \\ (P:\alpha) & \text{iff } P \neq A_n \end{cases}$	
$(\alpha)/_f\, \eta = \begin{cases} \mathsf{tt} & \text{iff } \alpha \in l_{A_n}(\eta) \\ \alpha & \text{otherwise} \end{cases}$	

Table 1.2 Structural definition of quotient $\varphi/_f\, \eta$ with η a node of A_n.

that contrary to the quotient defined in [16] we do not quotient the formula for any node and any clock region since we insert explicitly guards into the formula (and not only their truth value w.r.t. the regions). *This explain why such a quotient is much more compact and more efficient than the first one.*

Example 4 *The quotient of* $\mathsf{Mut}_{(1,2)}$ *by the automaton* V *is* $(X_0, \mathcal{D}')$ *with* $\mathcal{D}'$*:*

$$\begin{array}{lcl}
X_0 & \stackrel{def}{=} & (\neg P_1\!:\!CS \vee \neg P_2\!:\!CS) \wedge [=0]X_0 \wedge [:=1]X_1 \wedge [:=2]X_2 \wedge [\delta]\ X_0 \\
X_1 & \stackrel{def}{=} & (\neg P_1\!:\!CS \vee \neg P_2\!:\!CS) \wedge [=1]X_1 \wedge [:=1]X_1 \wedge [:=2]X_2 \wedge [\delta]\ X_1 \\
X_2 & \stackrel{def}{=} & (\neg P_1\!:\!CS \vee \neg P_2\!:\!CS) \wedge [=2]X_2 \wedge [:=1]X_1 \wedge [:=2]X_2 \wedge [\delta]\ X_2
\end{array}$$

□

Now, let π be the binary synchronization function completely specified by $\pi(\bullet, a) = a$ for all $a \in \mathcal{A}$ (i.e. the left argument to π is completely ignored). Then it is easy to see that any timed automaton A is strongly bisimilar [2] to the network $(nil|A)_\pi$ and hence can be proved to satisfy the same formula of L_ν. Using this observation, the quotient construction can be used to obtain alternative model–checking algorithms for L_ν as follows: $A \models \varphi \Leftrightarrow (nil|A)_\pi \models \varphi \Leftrightarrow nil \models \varphi/_\pi\, A$. Due to the projective nature of π it is clear that $\varphi/_\pi\, A$ contains no action modalities, and it is easy to build a special purpose model-checker for the simple automaton nil.

Note that the quotient definition would support the introduction of negation in the logic (by a simple propagation) and this would allow to use minimal fixpoint in the specification and then increase its expressive power. Nevertheless such an extension would require to adapt the simplification rules.

Simplification Strategies

We have seen in the previous section that repeated quotienting leads to an explosion in the formula. This phenomena was already observed in [16, 18, 15] and in the untimed case in [3]. An efficient way to avoid this explosion consists in applying *simplification rules after each quotient.*

First we need simplifications to tackle the explosion due to the parallel composition (occurring with the quotienting of the variables). This can be done by using the classical rules already used in the untimed case:

• *Boolean Simplification.* Formulas may be simplified using the following simple boolean equations and their duals: $\mathsf{ff} \wedge \varphi \equiv \mathsf{ff}$, $\mathsf{tt} \wedge \varphi \equiv \varphi$, $\langle a \rangle \mathsf{ff} \equiv \mathsf{ff}$, $\langle \delta \rangle \, \mathsf{ff} \equiv \mathsf{ff}$, $x \text{ in } \mathsf{ff} \equiv \mathsf{ff}$, $\langle a \rangle \varphi \wedge [a] \mathsf{ff} \equiv \mathsf{ff}$.

• *Constant Propagation.* Identifiers which are declared to either tt or ff may be removed while substituting their definitions in other equations.

• *Trivial Equation Elimination.* Equations of the form $X \stackrel{\text{def}}{=} [a]X$ are easily seen to have $X = \mathsf{tt}$ as solution. More generally, X can be replaced by tt when $\varphi[\mathsf{tt}/X]$ [3] (with $X \stackrel{\text{def}}{=} \varphi$) can be simplified to tt.

• *Equivalence Reduction.* If two identifiers X and Y are equivalent (i.e. are satisfied by the same timed transition systems) we may collapse them into a single identifier.

Secondly we have to add special simplifications to deal with the clocks constraints. Indeed the major problem in the real-time systems analysis consists in the treatment of time encoding: simplifying test or reducing the number of clocks used in the specification can be crucial to succeed in the verification of a system. We developed several strategies which have been evaluated with CMC. Here we present two simplifications which can be combined together and which lead to interesting results. The main idea is that we are interested in the truth value of a specification φ for extended configurations (s, u_0) (with s a network configuration) with all formula clocks equal to zero. Using this information, it is sometimes possible to simplify the tests inside φ:

• *Hitzone Reduction.* Consider the formula $\varphi = [a]\ (x > 1)$. Clearly φ is equivalent to $[a]\mathsf{ff}$ for any extended configuration (s, u_0) because no a-transition can modify the value of the formula clock x (which is still equal to 0 after the transition). A more complex example is: $\psi = [\delta]\ (x > 1 \ \Rightarrow\ [a]\, x < 1)$; for the same reason, ψ is equivalent to $[\delta]\ (x > 1 \ \Rightarrow\ [a]\, \mathsf{ff})$.

Given a specification φ, the principles of *Hitzone reduction* is to compute for any test t occurring in φ the clock constraints S_t (i.e. the set of time assignments for the formula clocks) which are effectively needed to evaluate the truth value of φ for any (s, u_0). Given this information (obtained by a fixpoint computation) we can simplify by tt (resp. ff) the tests t for which the truth value is true (resp. false) for every time assignment in S_t.

A more interesting example is the simplification of $\varphi = (X_0, \mathcal{D})$ with the following declaration $\mathcal{D}$:

$$\begin{array}{lcl} X_0 & \stackrel{\text{def}}{=} & [a]\,(x_1 \text{ in } X_1) \wedge [\delta]\ X_0 \\ X_1 & \stackrel{\text{def}}{=} & [b]\,(x_2 \text{ in } X_2) \wedge [\delta]\ X_1 \\ X_2 & \stackrel{\text{def}}{=} & x_1 \geq x_2 \wedge [\delta]\ X_2 \end{array}$$

An observation of the declaration shows that the truth value of X_2 is always considered from X_1 after a reset of x_2 or from its own definition. Then the test $x_1 \geq x_2$ in X_2 will always be equal to $\mathtt{tt}$, therefore X_2 becomes a trivial equation and therefore X_1 too. Finally X_0 can be reduced to $\mathtt{tt}$.

• *Constraint Reduction.* Another way to simplify the tests inside the formula is to compute the *constraint* associated to each variable X, that is the set of timed valuations which can satisfy the tests in its definition. For example, the constraint of $X \stackrel{\text{def}}{=} x < 1 \wedge \langle a \rangle (y < 2)$ is $[x < 1 \wedge y < 2]$: for any extended configuration (η, v, u) satisfying X, we have $u \models x < 1 \wedge y < 2$. Each L_ν operator has its own rule to propagate the constraint and a fixpoint computation is necessary to obtain the final constraint of any variable. For example, the constraint of a variable $X \stackrel{\text{def}}{=} x < 1 \wedge \ldots \wedge [\delta]\ X$ will be empty and X is then equal to $\mathtt{ff}$.

Finally it is possible to combine the Hitzone and the Constraint redusctions to simplify the tests inside the formula. Note that these two simplifications assume that the formula clocks and automata clocks are disjoint: we assume that the action transitions do not modify the value of formula clocks. Without these simplifications, we could use automata clocks in formula.

Example 5 *Recall Example 2. Without simplification (except boolean reductions) the quotient specification $mut_{12} /_{\!f}\, V /_{\!f}\, P_1 /_{\!f}\, P_2$ contains 18 identifiers. Applying the minimization strategies of the* CMC *tool we find that $mut_{12} /_{\!f}\, V /_{\!f}\, P_1 /_{\!f}\, P_2$ is equivalent to $\mathtt{tt}$. Then the property holds for the system.* □

∘ **Why it is efficient ?** Quotienting immediately followed by the various simplification strategies often allows to dismiss parts of the quotiented component which are irrelevant to the specific property in question. Also, the iterative strategy allows to deal with clock constraints after each quotienting i.e. before considering all clocks of the system, thus making simplification easier and more efficient (this will be clearly illustrated by Fischer's mutual exclusion protocol).

EXPERIMENTAL RESULTS WITH CMC

The compositional method has been implemented in the CMC [4] tool (written in C++) : it contains a command language and an environment to define networks of TA and L_ν specifications, and several functions to build quotient, to simplify formula (every reductions described in the previous sections has been implemented) ... Moreover it contains a `Check` procedure to decide if a formula holds for the *nil* process. It is also possible to verify if two timed automata without invariant are bisimilar.

In this section we present two examples verified with CMC. The first one is Fischer's mutual exclusion protocol, which is a benchmark in the timed verification area. This example is interesting because it shows both the power of compositional approach and its limits. Here we address the two classical versions of the protocol and we present a third one.
The second example is the train crossing example. Its purpose is to show that L_ν is an expressive specification language: we write a short specification including safety properties, bounded liveness property etc. Moreover, whereas Fischer's protocol has a symmetrical structure potentially making the compositional approach more efficient, the train crossing example has no special structure thus showing that the method is not limited to special systems but can be applied to arbitrary networks of timed automata.

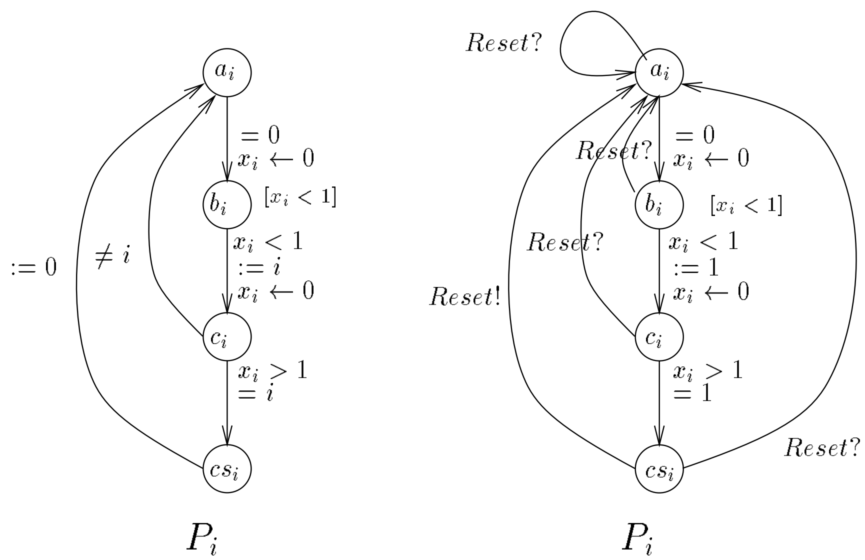
Fischer's mutual exclusion protocol

Fischer's mutual exclusion protocol described in the example 2 is a benchmark of the verification of timed systems. Consider n processes P_i, the problem is to verify that at any time at most one process can access its critical section: we want to verify that for any $i, j \in \{1, \ldots, n\}$ with $i \neq j$, $\mathsf{Mut}_{(i,j)}$ holds for the initial configuration of the system. Remember that $\mathsf{Mut}_{(i,j)} \stackrel{\text{def}}{=} \mathsf{ALWAYS}_{\mathcal{A}}(\neg P_i : cs_i \vee \neg P_j : cs_j)$.

In order to verify $\mathsf{Mut}_{(1,2)}$ efficiently, we need to carefully choose the order in which the processes are quotiented. In fact, since $\mathsf{Mut}_{(1,2)}$ only directly deals with the processes V, P_1 and P_2, we may hope that quotienting these three components will suffice for determining the truth value of the formula. Using CMC, we start by quotienting V, then P_1 and finally P_2. The simplification rules allow to deduce $\mathtt{tt}$! To evaluate the efficiency of the method we give the size of the specification (i.e. the number of variables). For example, we have the following computation [5] for $n = 100$:

Quotient by ...	V_{100}	P_1	P_2
Size of specification:	$202 \leadsto_s 102$	$404 \leadsto_s 202$	$800 \leadsto_s 0(\mathtt{tt})$

These results can be read as follows: The quotient of $\mathsf{Mut}_{(1,2)}$ by V_{100} gives a specification with 202 variables which are reduced to 102 after the simplifications, then the quotient by P_1 gives 404 variables which are reduced to 202, and finally the quotient of P_2 gives 800 variables which are reduced to the single formula $\mathtt{tt}$.
Of course a complete verification needs to do such a computation for any $\mathsf{Mut}_{(i,j)}$ but by choosing the right order for removing the components the previous result shows that the method can be successfully applied. Note that the



Complete version — Revisited complete version

Figure 1.3 The complete version of Fischer's mutual exclusion protocol and a variant.

compositional approach is very efficient in this case: other existing real-time verification tools cannot verify this protocol with more than 15 processes.

In fact the previous version of Fischer's mutual exclusion protocol is not the complete protocol. The complete version is cyclic and is described on the left part of figure 1.3. The main point is that a process using the critical section can leave it and then reset the variable V. This version is much more difficult to verify. Indeed if we quotient $\mathsf{Mut}_{(1,2)}$ (in which we add a term $[:= 0]X$ in the conjunction) by V, P_1 and P_2, we **cannot** deduce $\mathtt{tt}$ (contrary to what we claim in [15]) because at this step we know, thanks to the synchronization function, that other processes can perform the $:= 0$ action and we have no information about **when** they can perform it. Typically if P_3 was just a process able to reset V at any time, then P_2 could access the critical section at the same time as P_1. Then for this version of the protocol, we need to quotient all processes to finally deduce $\mathtt{tt}$. And in this case, we cannot avoid the explosion of the size of the specification and we are limited to small systems. We obtain the following computation for 6 processes:

V_6	P_1	P_2	P_3	P_4	P_5	P_6
$14 \rightsquigarrow_s 8$	$30 \rightsquigarrow_s 20$	$74 \rightsquigarrow_s 29$	$106 \rightsquigarrow_s 45$	$162 \rightsquigarrow_s 69$	$242 \rightsquigarrow_s 101$	$338 \rightsquigarrow_s 0$

Note that for this version, the newest BDD approach proposed in [7] for KRONOS can deal with 14 components. Nevertheless we can consider a third version of the Fischer protocol. It is described in the right part of the figure. The main idea is to see the reset of the variable as a reset of the protocol: when a process leaves the critical section, then it resets the variable and all processes receive

Train_1	Train_2	Controller	Counter	Gate	visible action
$App!$	$\bullet$	$App?Incr!$	$Incr?$	$\bullet$	App
$-$	$App!$	$App?Incr!$	$Incr?$	$\bullet$	App
$Exit!$	$\bullet$	$Exit?Reset!$	$Reset?$	$\bullet$	$Exit$
$Exit!$	$\bullet$	$Exit?Decr!$	$Decr?$	$\bullet$	$Exit$
$\bullet$	$Exit!$	$Exit?Reset!$	$Reset?$	$\bullet$	$Exit$
$\bullet$	$Exit!$	$Exit?Decr!$	$Decr?$	$\bullet$	$Exit$
$\bullet$	$\bullet$	$GoUp!$	$\bullet$	$GoUp?$	$GoUp$
$\bullet$	$\bullet$	$GoDown!$	$\bullet$	$GoDown?$	$GoDown$
i	$\bullet$	$\bullet$	$\bullet$	$\bullet$	i
$\bullet$	i	$\bullet$	$\bullet$	$\bullet$	i
$\bullet$	$\bullet$	$\bullet$	$\bullet$	i	i

Table 1.3 Synchronization table for (Train_1 | Train_2 | Controller | Counter | Gate).

this signal and go back to their initial state. To obtain such protocol it is sufficient to synchronize the *Reset* action over all processes. Note that the protocol is still cyclic and if such a change simplifies the protocol, it remains quite reasonable. In this case, we succeed [6] in applying the compositional approach:

Quotient by ...	V_{100}	P_1	P_2
Size of specification:	$202 \rightsquigarrow_s 102$	$404 \rightsquigarrow_s 301$	$1194 \rightsquigarrow_s 0$

The train crossing example

We now present a version of the classical railroad crossing example to illustrate the expressive power of L_ν. The network of timed automata (Train_1 | Train_2 | Controller | Counter | Gate$)_f$ in figures 1.4, 1.5 and 1.6 models a train crossing: The trains can send an $App!$ (resp. $Exit!$) message to a controller to inform it of their arrival in (resp. departure from) the section before (resp. after) the gate. The gate can receive orders from the controller and can change its position (open or close) in at most 10 seconds.
The controller uses the same strategy as the one presented in [8]: the idea is to count the number of trains which are staying between the *App* point and the *Exit* point and to use this information to decide if an $Exit!$ message has to be followed by a $GoUp!$ order. To model this strategy we use the automaton Counter whose nodes correspond to the possible values (bounded by 2). The synchronization function is given in table 1.3 and has to be read as follows: An *App* action of the system corresponds to a $App!$ action performed by one of the trains, a $App?Incr!$ action of the controller and a $Incr?$ action of the counter. Note that the distinction between $Reset?$ and $Decr?$ allows us to deduce whether there remains only a single train in the section.
In this example, we use name of nodes as atomic propositions.

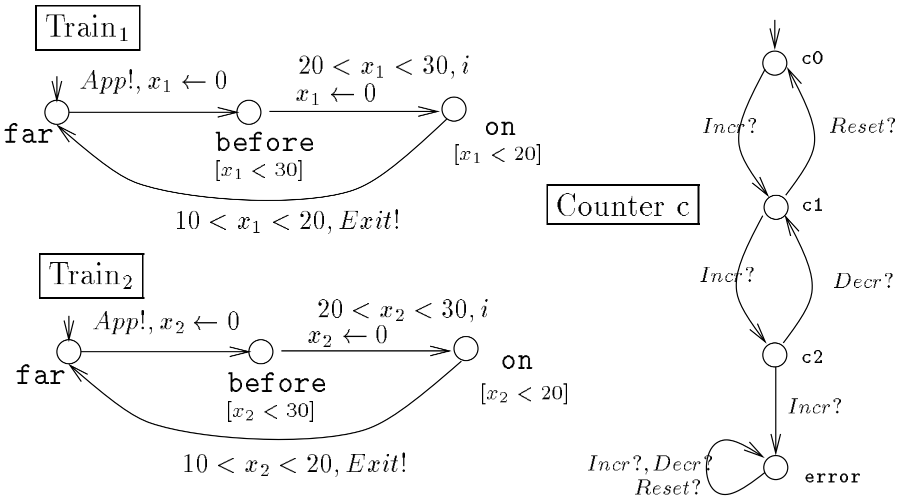


Figure 1.4 The trains and the counter.

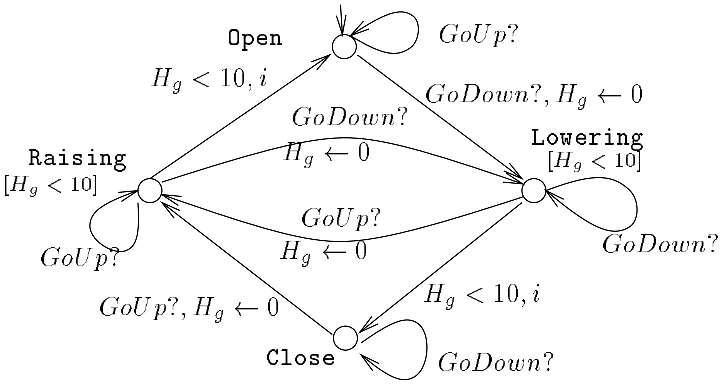


Figure 1.5 The gate.

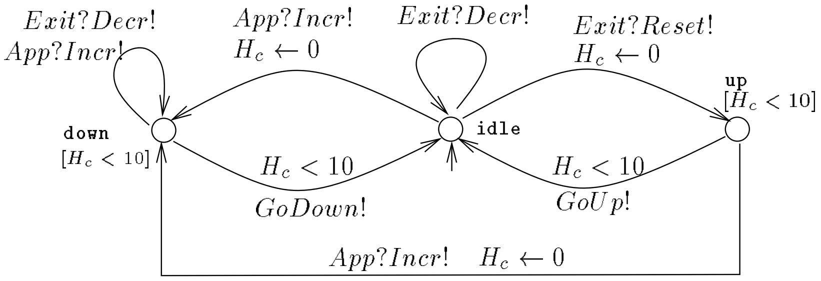


Figure 1.6 The controller.

Specifying with L_ν. An important point of CMC is the expressivity of its specification language, L_ν, which allows application specific properties to be expressed concisely. The aim of this part is to illustrate this point by specifying the train crossing example. Clearly we expect the following two properties to hold:

- "When a train is **on** the rail crossing, the gate is **close**". This property can be expressed by the formula:
  $\varphi_{\mathtt{safe}} = \mathsf{ALWAYS}_{\mathcal{A}}\Big((\mathrm{Train}_1 : \mathtt{on} \vee \mathrm{Train}_2 : \mathtt{on}) \;\Rightarrow\; \mathrm{Gate} : \mathtt{close}\Big)$

- "The gate is open in at most δ_l seconds after an **Exit** signal if no train approaches the railroad crossing inbetween". We can use the formula:

  $$\varphi_{\mathtt{live}}(\delta_l) = \mathsf{ALWAYS}_{\mathcal{A}}\Big[[Exit]\,\Big(z \;\mathsf{in}\; [(z<\delta_l)\mathsf{Until}_{\mathcal{A}\setminus App}\mathrm{Gate} : \mathtt{open}]\Big)\Big]$$

  Note that the restriction on the set of allowed actions in Until is useful in capturing the "if" part of the informal statement of the property.

However, in addition to the application specific properties above, we also want to ensure that the model is *realizable* or *physically valid* and possesses no unreasonable behavior:

- "There is no deadlock in the model: it is always possible to wait for some delay such that an action (App, $Exit$, i, $GoUp$, $GoDown$) is enabled". We can use $\varphi_{\mathtt{no\_dl}}$ with:

  $$\varphi_{\mathtt{no\_dl}} \stackrel{\mathrm{def}}{=} \mathsf{ALWAYS}_{\mathcal{A}}\Big[\langle\delta\rangle\Big(\langle i\rangle\mathtt{tt}\vee\langle App\rangle\mathtt{tt}\vee\langle Exit\rangle\mathtt{tt}\vee\langle GoDown\rangle\mathtt{tt}\vee\langle GoUp\rangle\mathtt{tt}\Big)\Big]$$

- Bounding the counter with 2 is sufficient: "the `error` state is not reachable". This can be expressed by $\varphi_{\mathtt{err}} \stackrel{\mathrm{def}}{=} \mathsf{ALWAYS}_{\mathcal{A}}(\neg\mathrm{Counter} : \mathtt{error})$.

- Actions $App?$, $Exit?$, $GoDown?$ and $GoUp?$ correspond to the reception of signals: they have to be enabled when a corresponding signal can be sent (unless this could prevent possible behaviors). The property clearly holds for App and $GoDown$ but it is unclear for $Exit?$ because there is no transition $Exit?$ from `up` node in the controller. We can use : $\varphi_{\mathtt{signal}} \stackrel{\mathrm{def}}{=} \mathsf{ALWAYS}_{\mathcal{A}}(\langle Exit!\rangle\,\mathtt{tt} \;\Rightarrow\; \langle Exit?\rangle\,\mathtt{tt})$.
  Note that to verify this property we have to add in the synchronization table that an $Exit!$ (resp. $Exit?$) signal can be performed by one of the train without synchronization (resp. by the controller with actions $Exit?Reset!$ or $Exit?Decr!$ with a synchronization over Counter actions). Being able to express such a property with L_ν is important because it gives a good help for the model construction.

Results. For our train example, we have the following results [7]:

	/Train$_1$	/Train$_2$	/Controller	/Counter	/Gate	`Check`	Res.
$\varphi_{\texttt{safe}}$	$12 \leadsto_s 7$	$30 \leadsto_s 27$	$65 \leadsto_s 59$	$48 \leadsto_s 40$	$116 \leadsto_s 0(\mathtt{tt})$	–	Yes
$\varphi_{\texttt{live}}(70)$	$17 \leadsto_s 17$	$55 \leadsto_s 55$	$141 \leadsto_s 127$	$96 \leadsto_s 76$	$216 \leadsto_s 0(\mathtt{tt})$	–	Yes
$\varphi_{\texttt{live}}(65)$	$17 \leadsto_s 17$	$55 \leadsto_s 55$	$141 \leadsto_s 127$	$96 \leadsto_s 76$	$216 \leadsto_s 77$	17 it.	No
$\varphi_{\texttt{no\_dl}}$	$14 \leadsto_s 14$	$66 \leadsto_s 50$	$154 \leadsto_s 112$	$98 \leadsto_s 65$	$237 \leadsto_s 83$	7 it.	Yes
$\varphi_{\texttt{err}}$	$6 \leadsto_s 6$	$27 \leadsto_s 27$	$65 \leadsto_s 59$	$48 \leadsto_s 0(\mathtt{tt})$	–	–	Yes
$\varphi_{\texttt{signal}}$	$11 \leadsto_s 10$	$40 \leadsto_s 33$	$79 \leadsto_s 68$	$61 \leadsto_s 42$	$124 \leadsto_s 51$	5 it.	Yes

The column "Check" contains the number of iterations (of the fixpoint computation) used by the `Check` procedure to decide if φ/Train$_1$/Train$_2$/Controller /Counter/Gate holds for the *nil* process when the formula has not already been reduced to $\mathtt{tt}$ or $\mathtt{ff}$ by the simplifications. The last column gives the final answer (Yes or No) for $S \models \varphi$.

Note that for $\varphi_{\texttt{err}}$ it is not necessary to quotient all automata: the analysis of a subpart of the system allows to verify that this formula holds for the system.

CONCLUSION

Since the introduction of the compositional method in [16], substantial improvements have been achieved. In particular, the tool CMC presented in this paper offers a collection of new simplification heuristics adding to the efficiency of the overall method. Partially based on the experimental results of this paper, our conclusion is that (1) the compositional method is efficient and not limited to special types of networks of timed automata and (2) the timed modal logic L_ν constitutes a convenient specification language.

The future development will concern the elaboration of new simplifications (based on partial evaluation of formula) and an optimization of the data structure to encode the clocks constraints.

Acknowledgments

We want to thank Béatrice Bérard and Philippe Schnoebelen for their helpful comments about this work.

Notes

1. The proof is omitted due to space limits but it is very similar to the proofs of [16, 18] but a long version of this paper including the proof is available by email.
2. We extend the notion of strong bisimulation to require that delay transitions are matched by delay transitions of equal time-quantity.
3. $\varphi[\mathtt{tt}/X]$ is the formula obtained by substituting all occurrences of X in φ with the formula $\mathtt{tt}$.
4. CMC can be obtained by sending an email to `fl@lsv.ens-cachan.fr`.

5. This has been done in few minutes on a SUN Ultra Sparc 1 with 64 Mb memory

6. This computation takes approximately one hour on a SUN Ultra Sparc 2 with 512 Mb.

7. All these results have been computed on a Sparc Ultra 1 with 64 Mb memory in few seconds except for $\varphi_{\mathtt{live}}(65)$ which needs approx 6 minutes.

References

[1] M. Abadi and L. Lamport. An Old-Fashioned Recipe for Real-Time. In *Proc. of REX Workshop "Real-Time: Theory and Practice",LNCS 600*, 1993.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

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

[4] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-Structures for the Verification of Timed Automata. In *Proc. of HART'97, LNCS 1201*, 1997.

[5] J. Bengtsson, W. O. David Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using uppaal. In *Proc. of the 8th International Conference on Computer-Aided Verification, LNCS 1102*, pages 244–256, 1996.

[6] A. Bouajjani, S. Stavros, and S. Yovine. On-the-Fly Symbolic Model Checking for Real-Time Systems. In *Proc. of the 18th IEEE Real-Time Systems Symposium, RTSS'97*. IEEE Computer Society Press, December 1997.

[7] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *Proc. of the 9th International Conference on Computer-Aided Verification, LNCS 1254*. Springer Verlag, 1997.

[8] N. Halbwachs, F. Maraninchi, and Y.E. Proy. The railroad crossing problem Modeling with Argos - Analysis with Polka. In *Proc. of 2nd European Workshops on Real-Time and Hybrid Systems*, 1995.

[9] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using uppaal. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, 1997.

[10] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[11] T.A. Henzinger, P.H. Ho, and H. Wong-Toi. A user guide to hytech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *Proceedings of TACAS'95*, volume 1019 of *LNCS*, pages 41–71. SPRINGER, 1995.

[12] T.A. Henzinger and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid systems II*, volume 999 of *LNCS*, pages 265–293. SPRINGER, 1995.

[13] T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In J.R. Abrial, E. Börger, and H. Langmaack, editors, *Formal methods for industrial applications. Specifying and programming the steam boiler control*, volume 1165 of *LNCS*, pages 265–282. SPRINGER, 1995.

[14] 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.

[15] K.J. Kristoffersen, F. Laroussinie, K.G. Larsen, P. Pettersson, and W. Yi. A Compositional Proof of a Real-Time Mutual Exclusion Protocol. In M. Bidoit and M. Dauchet, editors, *Proc. TAPSOFT'97, Lille, France, LNCS 1214*, pages 565–579. Springer-Verlag, April 1997.

[16] F. Laroussinie and K.G. Larsen. Compositional Model-Checking of Real Time Systems. In *Proc. CONCUR'95, Philadelphia, USA, LNCS 962*, pages 27–41. Springer-Verlag, August 1995.

[17] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium, RTSS'97*. IEEE Computer Society Press, December 1997.

[18] K. G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proc. of the 16th IEEE Real-Time Systems Symposium, RTSS'95*, 1995.

[19] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Journal of Software Tools for Technology Transfer*, 1(1/2), October 1997.

[20] O. Maler and S. Yovine. Hardware timing verification using KRONOS. In *Proc. 7th Israeli Conference on Computer Systems and Software Engineering*, Herzliya, Israel, June 1996.

[21] N. Shankar. Verification of Real-Time Systems Using PVS. In *Proc. of the 5th Int. Conf. on Computer Aided Verification, LNCS 697*, pages 210–224, 1993.

[22] S. Yovine. Kronos: A Verification Tool for real-Time Systems. *Journal of Software Tools for Technology Transfer*, 1(1/2), October 1997.