SRI International

CSL Technical Report SRI-CSL-02-03 • May 23, 2002

Bounded Model Checking for Timed Automata

Maria Sorea*

This research was supported by the National Science Foundation under grants CCR-00-82560 and CCR-00-86096.

*Also affiliated with: University of Ulm, Germany.

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (650) 859-2844

Abstract

Given a timed automaton M, a linear temporal logic formula φ, and a bound k, bounded model checking for timed automata determines if there is a falsifying path of length k to the hypothesis that M satisfies the specification φ. This problem can be reduced to the satisfiability problem for Boolean constraint formulas over linear arithmetic constraints. We show that bounded model checking for timed automata is complete, and we give lower and upper bounds for the length k of counterexamples. Moreover, we define bounded model checking for systems of timed automata in a compositional way.

Contents

List of Figures

Chapter 1

Introduction

Timed automata [AD94] are state-transition graphs augmented with a finite set of real-valued clocks. The clocks proceed at a uniform rate and constrain the times at which transitions may occur. Given a timed automaton and a property expressed in a timed logic such as TCTL [ACD90] or T_μ [HNSY94], model checking answers the question of whether or not the timed automaton satisfies the given formula. The fundamental graph-theoretic model checking algorithm by Alur, Courcoubetis, and Dill [ACD90] constructs a finite quotient, the so-called *region graph*, of the infinite state graph. Algorithms directly based on the explicit construction of such a partition are, however, unlikely to perform efficiently in practice, since the number of equivalence classes of states of the region graph grows exponentially with the largest time constant and the number of clocks that are used to specify timing constraints. Symbolic model checking algorithms are obtained by characterizing regions as Boolean combinations of linear inequalities over clocks [HNSY94]. Based on these algorithms, tools for verifying timed automata, such as Uppaal [LPY97], Kronos [DOTY96], HyTech [HHWT97], and Tempo [Sor01], have been developed.

As an alternative to classical model checking, the technique of bounded model checking has been recently introduced [CBRZ01]. Given a system M modeled as a state machine, a temporal logic specification φ, and a bound k, the bounded model checking (BMC) problem consists in searching for counterexamples of length k to the model checking problem $M \models \varphi$. The BMC problem for finite state models can be reduced to a propositional satisfiability problem,

and off-the-shelf propositional satisfiability (SAT) checkers are used to construct counterexamples from satisfying assignments to the propositional variables. It has been demonstrated that BMC is in many cases more effective in falsifying designs than traditional model checking techniques [CBRZ01, CFF+01]. In [dMRS02] the BMC paradigm has been extended to programs over infinite state space, and LTL formulas augmented with a decidable set of constraints. For an infinite state system M, a linear temporal logic formula with constraints φ, and a bound k, it has been illustrated how a Boolean constraint formula, $[\![M, \varphi]\!]_k$, can be constructed that is satisfiable if and only if there is a counterexample of length k for the model checking problem $M \models \varphi$. BMC for infinite state systems is sound, and for invariant properties also complete, but incomplete for the entire LTL logic.

The main contribution here is to show that BMC for timed automata is indeed complete for all LTL formulas with clock constraints. We describe how a timed automaton can be directly encoded into a Boolean constraint formula, without constructing the corresponding region graph. Our approach is compositional, in that Boolean constraint formulas encoding complex systems can be obtained by Boolean combinations of the encoding of the components. Obviously, this compositional approach reduces the size of the generated formula considerably. Moreover, we give lower and upper bounds for the length k of counterexamples that depend on the size of the LTL formula and the size of the region graph corresponding to the given timed automaton.

The paper is structured as follows. In Chapter 2 we provide some background information on Boolean constraints. Chapter 3 reviews the basic notions of timed automata. Chapter 4 presents the details of BMC for timed automata together with the completeness results. Lower and upper bounds for the length k of counterexamples are given. Chapter 5 illustrates BMC for networks, that is, parallel composition of timed automata, and shows how complex systems can be encoded into a Boolean constraint formula in a compositional way, without first computing the product automaton of the components. Finally, in Chapter 6 we present some experimental results using Fischer's mutual exclusion protocol as a benchmark, and draw conclusions.

Chapter 2

Background

A set of variables $V := \{x_1, \ldots, x_n\}$ is said to be typed if there are nonempty sets D_1 through D_n and a *type assignment* τ such that $\tau(x_i) = D_i$. For a set of typed variables V, a *variable assignment* is a function ν from variables $x \in V$ to an element of $\tau(x)$.

Let V be a set of typed variables and L be an associated logical language. A set of constraints in L is called a *constraint theory* $\mathcal{C}$ if it includes constants *true*, *false* and if it is closed under negation; a subset of $\mathcal{C}$ of constraints with free variables in $V' \subseteq V$ is denoted by $\mathcal{C}(V')$. For $c \in \mathcal{C}$ and ν an assignment for the free variables in c, the value of the predicate $[\![c]\!]_\nu$ is called the *interpretation* of c w.r.t. ν. Hereby, $[\![true]\!]_\nu$ ($[\![false]\!]_\nu$) is assumed to hold for all (for no) ν, and $[\![\neg c]\!]_\nu$ holds iff $[\![c]\!]_\nu$ does not hold. A set of constraints $C \subseteq \mathcal{C}$ is said to be *satisfiable* if there exists a variable assignment ν such that $[\![c]\!]_\nu$ holds for every c in C; otherwise, C is said to be *unsatisfiable*. Furthermore, a function $\mathcal{C}\text{-}sat(C)$ is called a $\mathcal{C}$-satisfiability solver if it returns $\perp$ if the set of constraints C is unsatisfiable and a satisfying assignment for C otherwise.

For a given theory $\mathcal{C}$, the set of *Boolean constraints* $\mathsf{Bool}(\mathcal{C})$ includes all constraints in $\mathcal{C}$ and it is closed under conjunction $\wedge$, disjunction $\vee$, and negation $\neg$. The notions of satisfiability, inconsistency, satisfying assignment, and satisfiability solver are homomorphically lifted to the set of Boolean constraints in the usual way. If $V = \{p_1, \ldots, p_n\}$ and the corresponding type assignment $\tau(p_i)$ is either true or false, then $\mathsf{Bool}(\{true, false\} \cup V)$ reduces to the usual notion of Boolean logic with propositional variables $\{p_1, \ldots, p_n\}$.

Chapter 3

Timed Automata

We review some basic notions of transition systems and timed automata. Timed automata, as introduced by Alur, Courcoubetis, and Dill [ACD90], are state-transition graphs augmented with a finite set of real-valued clocks. Given a set of clocks $Cl = \{x_1, \ldots, x_n\}$, a clock-valuation function $v : Cl \to \mathbb{R}_0^+$ assigns a (positive) real value to each clock. Clock constraints compare clock values with rational constants. Given a set Cl of clock variables (or simply *clocks*), x_1, x_2 arbitrary clocks, $\gamma \in \mathbb{Q}^{\geq 0}$, and $\sim \in \{\leq, \geq, <, >, =\}$, the set Φ of *clock (or timing) constraints* over Cl is defined by the grammar

$$g := \mathtt{tt} \mid \mathtt{ff} \mid x_1 \sim \gamma \mid x_1 - x_2 \sim \gamma \mid g_1 \wedge g_2.$$

For a positive integer d, $\Phi(d)$ is the finite subset of all timing constraints $x \sim \gamma$, $x - y \sim \gamma$, where $x, y \in Cl$, $\sim \in \{<, \leq, =, \geq, >\}$ and $\gamma \in \{0, \ldots, d\}$. Clock constraints over Cl are interpreted with respect to clock-valuation functions $v : Cl \to \mathbb{R}_0^+$. For a clock-valuation function v and a clock constraint g over Cl, we write $v \models g$ (to be read as "*v satisfies g*") to denote that according to the values given by v the constraint g evaluates to true. Formally, $v \models g$ is defined inductively over the syntactic structure of g, where $x_1, x_2 \in Cl$ are arbitrary clocks, $\gamma \in \mathbb{Q}^{\geq 0}$, and $\sim \in \{\leq, \geq, <, >, =\}$:

$$v \not\models \mathtt{ff} \qquad v \models \mathtt{tt} \qquad v \models x_1 - x_2 \sim \gamma \text{ iff } v(x_1) - v(x_2) \sim \gamma$$
$$v \models x_1 \sim \gamma \text{ iff } v(x_1) \sim \gamma \qquad v \models g_1 \wedge g_2 \text{ iff } v \models g_1 \text{ and } v \models g_2$$

For $\delta \in \mathbb{R}^{\geq 0}$, $v + \delta$ denotes the clock valuation that maps each clock $x \in Cl$ to the value $v(x) + \delta$. For a clock $x \in Cl$, $v[x := 0]$ denotes the clock valuation

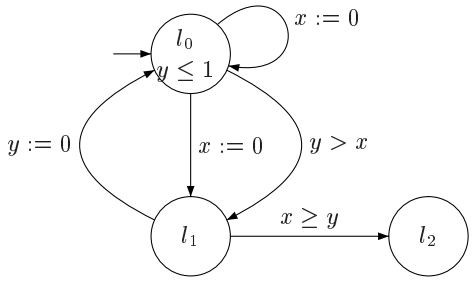

Figure 3.1: Example of a timed automaton (the *simple* example).

for Cl that maps x to the value 0 and leaves all the other clock values unchanged.

A *timed automaton* $\mathcal{S}$ is a tuple $\langle L, l_0, Cl, E, Inv \rangle$, where L is a nonempty finite set of locations, $l_0 \subseteq L$ is the initial location, and Cl is a finite set of clocks. $Inv : L \to \Phi$ assigns a set of downward closed clock constraints to each location L; the elements of $Inv(l)$ are the *invariants* for location l. $E \subseteq L \times \mathcal{P}(\Phi) \times \mathcal{P}(Cl) \times L$ is a finite set of edges. An edge $e = \langle l, g, r, l' \rangle$ represents a transition from location l to location l'. A transition may be fired only if the timing constraint (guard of the transition) g holds with respect to the current value of the clocks, and if the invariant of the target location is satisfied with respect to the modified value of the clocks. Firing a transition does not only change the current location but also resets the clocks in r to 0.

A timed automaton with three locations l_0, l_1, l_2 and two clocks x, y is displayed in Figure 3.1. The initial location is l_0, and transitions are decorated with both timing constraints and clock resets such as $x := 0$. The invariant for location l_0 is $y \leq 1$. Timing constraints that are *true* are omitted.

Alur, Courcoubetis, and Dill [ACD90] introduce the fundamental notion of clock regions, which partition the space of possible clock evaluation for a timed automaton into finitely many regions. For a timed automaton $\mathcal{S}$ with clocks Cl and largest constant d, occurring in any timing constraint of $\mathcal{S}$, a *clock region* is a set χ of clock valuations, such that for all timing constraints $g \in \Phi(d)$ and for any two v_1, $v_2 \in \chi$ it is the case that $v_1 \not\approx g$ if and only if $v_2 \not\approx g$. In this case we write $v_1 \equiv_{\mathcal{S}} v_2$. We will use $[v]$ to denote the clock region to which v belongs.

A *state* of a timed automaton $\mathcal{S}$ is a pair (l, v) where $l \in L$ is a location of $\mathcal{S}$ and v a clock valuation for Cl. An *initial state* is of the form (l_0, v_0) where l_0 denotes the initial state of $\mathcal{S}$ and v_0 maps all clocks in Cl to 0. We extend the satisfiability relation for clock constraints on states, as follows: for a state

(l, v) and a timing constraint g, $(l, v) \models g$ iff $v \models g$. A *timed step* is either a *delay step*, where time advances by some positive real-valued δ, or an instantaneous *state transition step*. For a timed automaton $\mathcal{S} = \langle L, l_0, Cl, E, Inv \rangle$, and $\delta \geq 0$, we say that the state $(l, v + \delta)$ is obtained from (l, v) by a *delay step* $(l, v) \xrightarrow{\delta} (l, v + \delta)$, if the invariant constraint $v + \delta \models Inv(l)$ holds. A *state transition step* $(l, v) \xrightarrow{g,r} (l', v')$ occurs if there exists an edge $\langle l, g, r, l' \rangle$, and $v \models g$, $v' = v[r := 0]$, and $v' \models Inv(l')$. The union of delay and state transition steps defines the timed transition relation $\Rightarrow$ of a timed automaton $\mathcal{S}$. Now, a *path* π is an infinite sequence of states $(l_0, v_0), (l_1, v_1), \ldots$ such that $(l_i, v_i) \Rightarrow (l_{i+1}, v_{i+1}), \forall i \geq 0$.

Chapter 4

System Verification

In presenting the details of BMC for timed automata together with the completeness results, we assume as given a solvable constraint theory $\mathcal{C}$ that includes the clock constraints Φ, and constraints of the form $x' - x = y' - y$, where $x, x', y, y' \in Cl$ are clock variables. To make this paper as self-contained as possible, we recall some notions and definitions from [dMRS02]. For the simplicity of the presentation we consider only timed automata that are nonzeno. A complete BMC procedure for timed automata, however, requires an explicit encoding of nonzenoness such as, for example, the one given in [MRS02].

Definition 1 ($\mathcal{C}$-*Programs*) Typed variables in $V := \{x_1, \ldots, x_n\}$ are also called *state variables*, and a *program state* is a variable assignment over V. A pair $\langle I, T \rangle$ is a $\mathcal{C}$-program over V if $I \in \mathsf{Bool}(\mathcal{C}(V))$ and $T \in \mathsf{Bool}(\mathcal{C}(V \cup V'))$, where V' is a primed, disjoint copy of V. I is used to restrict the set of initial program states, and T specifies the transition relation between states and their successor states. The set of $\mathcal{C}$-*programs* over V is denoted by $\mathsf{Prg}(\mathcal{C}(V))$.

The semantics of a program P is given in terms of a *transition system* M in the usual way, and, by a slight abuse of notation, we sometimes write M for both the program and its associated transition system.

A timed automaton $\mathcal{S} = \langle L, l_0, Cl, E, Inv \rangle$ can easily be described in terms of a program with linear arithmetic constraints over states $(at, x_1, \ldots, x_n)$, where at is interpreted over the set L of locations and the clock variables $x_1, \ldots, x_n \in Cl$ are interpreted over $\mathbb{R}_0^+$.

Definition 2 Given a timed automaton $\mathcal{S} = \langle L, l_0, Cl, E, Inv \rangle$, with $Cl = \{x_1, \ldots, x_n\}$ the set of clocks. $\mathcal{S}$ can be defined as a $\langle I, T \rangle$ program in $\mathsf{Prg}(\mathcal{C}(V))$ over the set $V = \{at, x_1, \ldots, x_n, at', x'_1, \ldots, x'_n\}$ as follows.

- Definition of the initial state l^0

$$I := (at = l_0 \ \wedge \ x_1 = 0 \ \wedge \ \ldots \ \wedge \ x_n = 0).$$

- Definition of a state transition step corresponding to $e = \langle l, g, r, l' \rangle \in E$

$$\tilde{T}(e) := (at = l \ \wedge \ g \ \wedge \ x'_1 = z_1 \ \wedge \ \ldots \ \wedge \ x'_n = z_n \ \wedge \ at' = l')$$

  where $z_i = 0$ if $x_i \in r$; otherwise $z_i = x_i$.

- Definition of delay steps ($Inv(\mathcal{S})$ is the set of all locations that have an invariant different from *true*.)

$$\begin{aligned} delay \quad := \quad \exists \delta \geq 0. \ (& \bigwedge_{l \in Inv(\mathcal{S})} (at = l \Rightarrow Inv(l)(x'_1, \ldots, x'_n)) \\ & \wedge \ (at' = at) \\ & \wedge \ (x'_1 = x_1 + \delta) \ \wedge \ \ldots \ \wedge \ (x'_n = x_n + \delta)). \end{aligned}$$

  The state formula $Inv(l)(x'_1, \ldots, x'_n)$ is obtained from the invariant of location l, $Inv(l)$, by replacing the variables $x_1, \ldots, x_n$ in the constraints of $Inv(l)$ by the primed variables $x'_1, \ldots, x'_n$.

- Definition of the transition relation T ($\otimes$ denotes the exclusive or connective)

$$T := \otimes_{e \in E} \tilde{T}(e) \otimes delay.$$

The timed automaton depicted in Figure 3.1, for example, is expressed in terms of the program $\langle I, T \rangle$ over states (at, x, y), where at is interpreted over the set of locations $\{l_0, l_1, l_2\}$, and the clock variables x, y are interpreted over $\mathbb{R}_0^+$. Initially, the program is in location l_0 and the value of the clocks x, y is equal to 0. The transitions are encoded by a conjunction of constraints over the current state variables at, x, y and the next state variables at', x', y'.

$$\begin{aligned} I(at, x, y) \quad &:= \quad (at = l_0 \ \wedge \ x = 0 \ \wedge \ y = 0) \\ T(at, x, y, at', x', y') \quad &:= \quad (at = l_0 \ \wedge \ x' = 0 \ \wedge \ y' = y \ \wedge \ at' = l_0) \otimes \\ & \qquad (at = l_0 \ \wedge \ x' = 0 \ \wedge \ y' = y \ \wedge \ at' = l_1) \otimes \end{aligned}$$

$$
\begin{aligned}
&(at = l_0 \ \wedge \ y > x \ \wedge \ x' = x \ \wedge \ y' = y \ \wedge \ at' = l_1) \otimes \\
&(at = l_1 \ \wedge \ y' = 0 \ \wedge \ x' = x \ \wedge \ at' = l_0) \otimes \\
&(at = l_1 \ \wedge \ x \geq y \ \wedge \ x' = x \ \wedge \ y' = y \ \wedge \ at' = l_2) \otimes \\
&\mathit{delay}(at, x, y, at', x', y')
\end{aligned}
$$

The delay steps are encoding as

$$
\begin{aligned}
&\mathit{delay}(at, x, y, at', x', y') := \\
&\qquad \exists \delta \geq 0.\ ((at = l_0 \Rightarrow y' \leq 1) \ \wedge \ (at' = at) \ \wedge (x' = x + \delta) \ \wedge \ (y' = y + \delta)).
\end{aligned}
$$

The above formula is not contained in $\mathsf{Bool}(\mathcal{C})$, since the definition of *delay* contains an existential quantifier. After performing quantifier elimination we obtain

$$
\begin{aligned}
&\mathit{delay}(at, x, y, at', x', y') \ := \\
&\qquad ((at = l_0 \Rightarrow y' \leq 1) \ \wedge \ (x' - x \geq 0) \ \wedge \ (y' - y = x' - x) \wedge \ (at' = at)).
\end{aligned}
$$

The formulas of the *constraint linear temporal logic* $\mathsf{LTL}(\mathcal{C})$ are linear-time temporal logic formulas with the usual "until" and "release" operators, and constraints $c \in \mathcal{C}$ as atoms. Note that only constraints in Φ are allowed.

$$
\varphi ::= \mathit{true} \mid \mathit{false} \mid c \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathbf{U} \, \varphi_2 \mid \varphi_1 \, \mathbf{R} \, \varphi_2
$$

The derived operators $\mathbf{F}\,\varphi = \mathit{true}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\,\varphi = \mathit{false}\,\mathbf{R}\,\varphi$ denote "eventually φ" and "globally φ" . Our logic does not contain a next-step operator. The main interest in removing the next-step operator stems from the fact that we do not want to distinguish between one delay step of duration, say, 1 and two subsequent delay steps of durations 2/5 and 3/5, since these traces are considered to be observationally equivalent. Logics without an explicit next-step operator have also been considered, for example, by Alur [Alu91], by Henzinger, Nicollin, Sifakis, and Yovine [HNSY94], and by Dams [Dam96]. Given a program $M \in \mathsf{Prg}(\mathcal{C})$ and a path π in M, the satisfiability relation $M, \pi \models \varphi$ for an $\mathsf{LTL}(\mathcal{C})$ formula φ is given in the usual way with the notable exception of the case of constraint formulas c. In this case, $M, \pi \models c$ if and only if c holds in the start state of π.

Definition 3 (Semantics of $\mathsf{LTL}(\mathcal{C})$) Given a program $M \in \mathsf{Prg}(\mathcal{C}(V))$ over the set of typed variables V, a path π in the transition system associated with

M, and a formula $\varphi \in \mathsf{LTL}(\mathcal{C}(V))$, the satisfiability relation $M, \pi \models \varphi$ is defined inductively over the syntax of φ.

$$
\begin{array}{rll}
M, \pi \models \mathit{true} & & \\
M, \pi \not\models \mathit{false} & & \\
M, \pi \models c & \text{iff} & \pi(0) \approxeq c \\
M, \pi \models \varphi_1 \wedge \varphi_2 & \text{iff} & M, \pi \models \varphi_1 \text{ and } M, \pi \models \varphi_2 \\
M, \pi \models \varphi_1 \vee \varphi_2 & \text{iff} & M, \pi \models \varphi_1 \text{ or } M, \pi \models \varphi_2 \\
M, \pi \models \varphi_1 \,\mathbf{U}\, \varphi_2 & \text{iff} & \exists i. M, \pi^i \models \varphi_1 \text{ and } \forall j < i. M, \pi^j \models \varphi_2 \\
M, \pi \models \varphi_1 \,\mathbf{R}\, \varphi_2 & \text{iff} & \forall i. M, \pi^i \models \varphi_1 \text{ or } \exists j < i. M, \pi^j \models \varphi_2
\end{array}
$$

Assuming the notation above, the *$\mathcal{C}$-model checking problem* $M \models \varphi$ holds iff for all paths $\pi = s_0, s_1, \ldots$ in M with $s_0 \in I$ it is the case that $M, \pi \models \varphi$.

The following lemma states that the logic $\mathsf{LTL}(\mathcal{C})$ preserves bisimulation.

Lemma 1 Given a program M with a finite bisimulation M' (i.e., $M \approx M'$), and a formula $\varphi \in \mathsf{LTL}(\mathcal{C})$; then $M \models \varphi$ iff $M' \models \varphi$.

Proof. The proof follows by induction over the structure of φ. The cases $\varphi = \mathit{true}$ and $\varphi = \mathit{false}$ are trivial.

$\boxed{\varphi = c}$ Assume $M, \pi \models \varphi$ for all paths $\pi = (l_0, v_0), (l_1, v_1), \ldots$ in M. Then by Definition 3 $M, \pi \models c$ iff $(l_0, v_0) \approxeq c$. From $(l_0, v_0) \approxeq c$ by the definition of clock regions it follows that $(l_0, [v_0]) \approxeq c$, where $[v_0]$ denotes the clock region of M with $v_0 \in [v_0]$. Again by Definition 3, we obtain that $M', [\pi] \models c$, where $[\pi] = (l_0, [v_0]), (l_1, [v_1]), \ldots$.

$\boxed{\varphi = \varphi_1 \,\mathbf{U}\, \varphi_2}$ Assume $M, \pi \models \varphi$ for all paths $\pi = (l_0, v_0), (l_1, v_1), \ldots$ in M. Then, by Definition 3 there exists $i \geq 0$ such that $M, \pi^i \models \varphi_1$ and $M, \pi^j \models \varphi_2$, $\forall j < i$. From the fact that M and M' are bisimilar, we can construct a path $\pi' = (l_0, [v_0]), (l_1, [v_1]), \ldots$, such that $v_i \in [v_i]$ for all $i \geq 0$. By induction hypothesis, $M', \pi'^i \models \varphi_1$ and $M', \pi'^j \models \varphi_2$, and therefore by Definition 3, $M', \pi' \models \varphi$. Since $\pi \approx \pi'$ for all paths π in M and π' in M', it follows that $M' \models \varphi$.

$\boxed{\varphi = \varphi_1 \,\mathbf{R}\, \varphi_2}$ Assume $M, \pi \models \varphi$ for all paths $\pi = (l_0, v_0), (l_1, v_1), \ldots$ in M. Then, by Definition 3 for all $i \geq 0$, $M, \pi^i \models \varphi_1$ or there exists $j < i$ such that $M, \pi^j \models \varphi_2$. From the fact that M and M' are bisimilar, we can construct a

path $\pi' = (l_0, [v_0]), (l_1, [v_1]), \ldots$, such that $v_i \in [v_i]$ for all $i \geq 0$. By induction hypothesis, $M', \pi'^i \models \varphi_1$ for all $i \geq 0$, or $M', \pi'^j \models \varphi_2$ for some $j < i$. Thus, by Definition 3, $M', \pi' \models \varphi$. Since $\pi \approx \pi'$ for all paths π in M and π' in M', it follows that $M' \models \varphi$.

$\boxed{\varphi = \varphi_1 \ \wedge \ \varphi_2}$ Follows by induction hypothesis.

$\boxed{\varphi = \varphi_1 \ \vee \ \varphi_2}$ Follows by induction hypothesis. □

Now, given a bound k, a program $M \in \mathsf{Prg}(\mathcal{C}(V))$ and a formula $\varphi \in \mathsf{LTL}(\mathcal{C})$ we consider the problem of constructing a formula $[\![M, \varphi]\!]_k \in \mathsf{Bool}(\mathcal{C}(V))$, which is satisfiable if and only if there is a counterexample of length k for the $\mathcal{C}$-model checking problem $M \models \varphi$. This construction proceeds as follows.

1. Definition of $[\![M]\!]_k$ as the unfolding of the program M up to step k from initial states (this requires k disjoint copies of V).
2. Translation of $\neg\varphi$ into a corresponding Büchi automaton $\mathcal{B}_{\neg\varphi}$ whose language of accepting words consists of the satisfying paths of $\neg\varphi$.
3. Encoding of the transition system for $\mathcal{B}_{\neg\varphi}$ and the Büchi acceptance condition as a Boolean formula, say $[\![\mathcal{B}]\!]_k$.
4. Forming the conjunction $[\![M, \varphi]\!]_k := [\![\mathcal{B}]\!]_k \wedge [\![M]\!]_k$.
5. A satisfying assignment for the formula $[\![M, \varphi]\!]_k$ induces a counterexample of length k for the model checking problem $M \models \varphi$.

Definition 4 (Encoding of $\mathcal{C}$-*Programs*) The encoding $[\![M]\!]_k$ of the kth unfolding of a *$\mathcal{C}$-program* $M = \langle I, T \rangle$ in $\mathsf{Prg}(\mathcal{C}(\{x_1, \ldots, x_n\}))$ is given by the Boolean constraint formula $[\![M]\!]_k$.

$$\begin{aligned} I_0(x[0]) &:= I\langle \{x_i \mapsto x_i[0] \mid x_i \in V\} \rangle \\ T_j(x[0], \ldots, x[k]) &:= T\langle \{x_i \mapsto x_i[j] \mid x_i \in V\} \cup \{x_i' \mapsto x_i[j+1] \mid x_i \in V\} \rangle \\ [\![M]\!]_k &:= I_0(x[0]) \wedge \bigwedge_{j=0}^{k-1} T_j(x[j], x[j+1]) \end{aligned}$$

where $\{x_i[j] \mid 0 \leq j \leq k\}$ is a family of typed variables for encoding the state of variable x_i in the jth step, $x[j]$ is used as an abbreviation for $x_1[j] \ldots, x_n[j]$, and $T\langle x_i \mapsto x_i[j] \rangle$ denotes simultaneous substitution of the x_i by $x_i[j]$ in formula T.

A two-step unfolding of the *simple* program in Figure 3.1, for example, is encoded by $[\![simple]\!]_2 := I_0 \wedge T_0 \wedge T_1$ $(*)$.

$$
\begin{aligned}
I_0 \quad := \quad & (at[0] = l_0 \wedge x[0] = 0 \wedge y[0] = 0) \\
T_0 \quad := \quad & (at[0] = l_0 \wedge x[1] = 0 \wedge y[1] = y[0] \wedge at[1] = l_0) \otimes \\
& (at[0] = l_0 \wedge x[1] = 0 \wedge y[1] = y[0] \wedge at[1] = l_1) \otimes \\
& (at[0] = l_0 \wedge y[0] > x[0] \wedge x[1] = x[0] \wedge y[1] = y[0] \wedge at[1] = l_1) \otimes \\
& (at[0] = l_1 \wedge y[1] = 0 \wedge x[1] = x[0] \wedge at[1] = l_0) \otimes \\
& (at[0] = l_1 \wedge x[0] \geq y[0] \wedge x[1] = x[0] \wedge y[1] = y[0] \wedge at[1] = l_2) \otimes \\
& ((at[0] = l_0 \Rightarrow y[1] \leq 1) \wedge (x[1] - x[0] \geq 0) \wedge \\
& \quad (y[1] - y[0] = x[1] - x[0]) \wedge (at[1] = at[0])) \\
T_1 \quad := \quad & (at[1] = l_0 \wedge x[2] = 0 \wedge y[2] = y[1] \wedge at[2] = l_0) \otimes \\
& (at[1] = l_0 \wedge x[2] = 0 \wedge y[2] = y[1] \wedge at[2] = l_1) \otimes \\
& (at[1] = l_0 \wedge y[1] > x[1] \wedge x[2] = x[1] \wedge y[2] = y[1] \wedge at[2] = l_1) \otimes \\
& (at[1] = l_1 \wedge y[2] = 0 \wedge x[2] = x[1] \wedge at[2] = l_0) \otimes \\
& (at[1] = l_1 \wedge x[1] \geq y[1] \wedge x[2] = x[1] \wedge y[2] = y[1] \wedge at[2] = l_2) \otimes \\
& ((at[1] = l_0 \Rightarrow y[2] \leq 1) \wedge (x[2] - x[1] \geq 0) \wedge \\
& \quad (y[2] - y[1] = x[2] - x[1]) \wedge (at[2] = at[1]))
\end{aligned}
$$

A difference between our approach and the BMC method presented in [CBRZ01] consists in the encoding of the LTL formulas. While in [CBRZ01] LTL formulas are translated directly into propositional formulas, we use Büchi automata for the encoding. This simplifies substantially the notations and the proofs. The translation of linear temporal logic formulas into a corresponding Büchi automaton is well studied in the literature (e.g., [GPVW95]) and does not require additional explanation. Notice, however, that the translation of $\mathsf{LTL}(\mathcal{C})$ formulas yields Büchi automata with $\mathcal{C}$-*constraints* as labels. Both the resulting transition system and the bounded acceptance test based on the detection of reachable cycles with at least one final state can easily be encoded as Boolean constraint formulas [dMRS02].

Definition 5 (Encoding of Büchi Automata) Let $V = \{x_1, \ldots, x_n\}$ be a set of typed variables, $\mathcal{B} = \langle \Sigma, Q, \Delta, Q^0, F \rangle$ be a Büchi automaton with labels Σ in $\mathsf{Bool}(\mathcal{C})$, and pc be a variable (not in V), which is interpreted over the finite

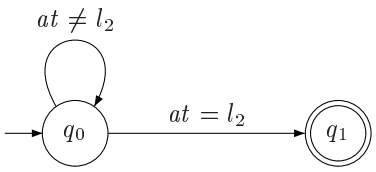

Figure 4.1: Automaton for $\mathbf{F}\,(at = l_2)$.

set of locations Q of the Büchi automaton. For a given integer k, we obtain, as in Definition 4, families of variables $x_i[j]$, $pc[j]$ $(1 \leq i \leq n,\ 0 \leq j \leq k)$ for representing the jth state of $\mathcal{B}$ in a run of length k. Furthermore, the transition relation of $\mathcal{B}$ is encoded in terms of the *$\mathcal{C}$-program* $\mathcal{B}_M$ over the set of variables $\{pc\} \cup V$, and $[\![\mathcal{B}_M]\!]_k$ denotes the encoding of this program as in Definition 4. Now, given an encoding of the acceptance condition

$$acc(\mathcal{B})_k \;:=\; \bigvee_{j=0}^{k-1} \Big(pc[k] = pc[j] \wedge \bigwedge_{v=1}^{n} x_v[k] = x_v[j] \wedge \Big(\bigvee_{l=j+1}^{k} \bigvee_{f \in F} pc[l] = f \Big) \Big)$$

the k-th unfolding of $\mathcal{B}$ is defined by $[\![\mathcal{B}]\!]_k := [\![\mathcal{B}_M]\!]_k \wedge acc(\mathcal{B})_k$.

Note that, as illustrated in [dMRS02], whenever an $\mathsf{LTL}(\mathcal{C})$ formula does not contain any release operators (**R**-free formula) it suffices to build an ordinary automaton over finite words instead of a Büchi automaton. Every **R**-free formula can be translated into an automaton over finite words that accepts a prefix of all infinite paths satisfying the given formula.

Definition 6 Given an automaton $\mathcal{B}$ over finite words and the notation as in Definition 5, the encoding of the k-ary unfolding of $\mathcal{B}$ is given by $[\![\mathcal{B}_M]\!]_k \wedge acc(\mathcal{B})_k$ with the acceptance condition

$$acc(\mathcal{B})_k \;:=\; \bigvee_{j=0}^{k} \bigvee_{f \in F} pc[j] = f \;.$$

Consider the problem of finding a counterexample of length $k = 2$ to the hypothesis that our running example in Figure 3.1 satisfies $\mathbf{G}\neg(at = l_2)$, that is, the timed automaton never reaches location l_2. The negated property $\mathbf{F}\,(at = l_2)$ is an **R**-free formula, and the corresponding automaton $\mathcal{B}$ over finite words is displayed in Figure 4.1. This automaton is translated, according to Definition 6, into the formula

$$[\![\mathcal{B}]\!]_2 := I(\mathcal{B}) \wedge T_0(\mathcal{B}) \wedge T_1(\mathcal{B}) \wedge acc(\mathcal{B})_2 \;. \qquad (**)$$

The variables $pc[j]$ and $x[j]$ $(j = 0, 1, 2)$ are used to represent the first three states in a run.

$$\begin{aligned}
I(\mathcal{B}) &:= (pc[0] = q_0) \\
T_0(\mathcal{B}) &:= (pc[0] = q_0 \wedge \neg(at[0] = l_2) \wedge pc[1] = q_0) \otimes \\
&\quad\ (pc[0] = q_0 \wedge at[0] = l_2 \wedge pc[1] = q_1) \\
T_1(\mathcal{B}) &:= (pc[1] = q_0 \wedge \neg(at[1] = l_2) \wedge pc[2] = q_0) \otimes \\
&\quad\ (pc[1] = q_0 \wedge at[1] = l_2 \wedge pc[2] = q_1) \\
acc(\mathcal{B})_2 &:= (pc[0] = q_1 \vee pc[1] = q_1 \vee pc[2] = q_1)
\end{aligned}$$

The bounded model checking problem $[\![simple]\!]_2 \wedge [\![\mathcal{B}]\!]_2$ for the *simple* program is obtained by conjoining the formulas $(*)$ and $(**)$. Using the BMC procedure over linear arithmetic constraints, one finds the counterexample

$$(l_0, x = 0, y = 0) \rightarrow (l_1, x = 0, y = 0) \rightarrow (l_2, x = 0, y = 0)$$

of length 2. Counterexamples for timed property, such as $\mathbf{G}\,(at = l_1 \Rightarrow x > y)$, can also be found by the BMC procedure.

The following two theorems are due to [dMRS02].

Theorem 1 (Soundness) Let $M \in \mathsf{Prg}(\mathcal{C})$ and $\varphi \in \mathsf{LTL}(\mathcal{C})$. If there exists a natural number k such that $[\![M, \varphi]\!]_k$ is satisfiable, then $M \not\models \varphi$.

Theorem 2 (Completeness for Finite State Systems) Let M be a *$\mathcal{C}$-program* with a finite set of reachable states, φ be an $\mathsf{LTL}(\mathcal{C})$ formula φ, and k be a given bound; then $M \not\models \varphi$ implies $\exists k \in I\!N.\ [\![M, \varphi]\!]_k$ is satisfiable.

In general, BMC over infinite domains is not complete. Consider, for example, the model checking problem $M \models \varphi$ for the program $M = \langle I, T\rangle$ over the variable $V = \{x\}$ with $I = (x = 0)$ and $T = (x' = x + 1)$ and the formula $\varphi = \mathbf{F}\,(x < 0)$. M can be seen as a one-counter automaton, where initially the value of the counter x is 0, and with every transition the value of x is increased with 1. Obviously, it is the case that $M \not\models \varphi$, but there exists no $k \in I\!N$, such that the formula $[\![M, \varphi]\!]_k$ is satisfiable. Since $\neg\varphi$ is not an $\mathbf{R}$-free formula, the encoding of the Büchi automaton $\mathcal{B}_k$ must contain, by Definition 5, a finite accepting cycle, described by $pc[k] = pc[0] \wedge x[k] = x[0]$ or $pc[k] = pc[1] \wedge x[k] = x[1]$ and so on. Such a cycle, however, does not exist, since the program M contains only one noncycling, infinite path, where the value of x increases in every step, that is, $x[i + 1] = x[i] + 1$, for all $i \geq 0$.

Theorem 3 (Completeness for Timed Automata) Let M be a timed automaton defined as a *$\mathcal{C}$-program* over a set of state variables $V = \{x_1, \ldots, x_n\}$, and φ be a formula in $\mathsf{LTL}(\mathcal{C})$; then

$$M \not\models \varphi \text{ implies } \exists k.\, [\![M, \varphi]\!]_k \text{is satisfiable.}$$

Proof. Let M' be the finite region graph corresponding to M, also defined as a $\mathcal{C}$-program over the set of state variables V. From $M \not\models \varphi$, it follows by Lemma 1, that $M' \not\models \varphi$. Let

$$[\![M', \varphi]\!]_k := [\![\mathcal{B}]\!]_k \wedge [\![M']\!]_k$$

be the bounded model checking problem for M' and φ. Since M' is finite, by Theorem 2 there exists a k such that $[\![M', \varphi]\!]_k$ is satisfiable. It remains to show, that if $[\![M', \varphi]\!]_k$ is satisfiable then also $[\![M, \varphi]\!]_k$ is satisfiable. From $[\![M', \varphi]\!]_k$ satisfiable it follows that $[\![M']\!]_k$ and $[\![\mathcal{B}]\!]_k$ are satisfiable. By Definition 4

$$[\![M']\!]_k := I'_0(x[0]) \wedge \bigwedge_{j=0}^{k-1} T'_j(x[j], x[j+1])$$

where the state formula $I'_0(x[0])$ encodes the initial state $(l_0, [v_0])$, and the formula $T'_j(x[j], x[j+1])$ defines the transition relation. Obviously, the formula $I'_0(x[0])$ is equivalent to the state formula $I_0(x[0])$, which describes the initial state (l_0, v_0) of the program M. Let $\pi' = s'_0, s'_1, \ldots, s'_{k-1}$, where $s'_i = (l'_i, [v'_i])$ be a k-path in M'. In [TY01] it has been shown that the region equivalence is a bisimulation relation. Since M and M' are bisimilar, it follows that there exists a k-path $\pi = s_0, s_1, \ldots, s_{k-1}$ in M, where $s_i = (l_i, v_i)$ such that $l_i = l'_i$ and $v_i \in [v'_i]$. Therefore, we can unfold M up to step k, in a manner similar to the unfolding of M', such that $[\![M]\!]_k$ and $[\![M']\!]_k$ are equisatisfiable. □

Lower bounds for the length k of counterexamples can be found by examining the structure of the Büchi automaton for a given $\mathsf{LTL}(\mathcal{C})$ formula. A lower bound is given by the length of the shortest path from the initial state to a final/accepting state of the automaton. For a timed automaton M with c the largest constant appearing in the guards and invariants of M, and t the number of clocks, an upper bound for k is given by

$$k \leq 2^{O(t \log(ct))} \cdot 2^{O(|\varphi|)}$$

where $2^{O(t \log(ct))}$ denotes the number of states in the region graph of M [Alu99].

Corollary 1 Let M be a timed automaton with c the largest constant appearing in the guards and invariants of M, and t the number of clocks. Further, let φ be a formula in $\mathsf{LTL}(\mathcal{C})$. If $k = 2^{O(t \log(ct))} \cdot 2^{O(|\varphi|)}$ then $M \models \varphi$ iff $[\![M, \varphi]\!]_k$ is unsatisfiable.

Chapter 5

BMC for Networks of Timed Automata

Complex systems are modeled as *networks of timed automata*, that is, parallel composition of timed automata. Given two timed automata A_1 and A_2, for defining synchronization on same events, we assume two finite alphabets Σ_1 and Σ_2, whose elements are used to label the transitions of A_1, respectively A_2. An edge of an automaton over an input alphabet Σ is now a tuple $e = \langle l, a, g, r, l' \rangle$. The product $A_1 \| A_2$ is defined in the obvious way [Alu99]. The locations of the product automaton are pairs of locations of its constituent automata. The invariant of a new location consists of the conjunction of the invariants of the component locations. Symbols that belong to both alphabets are used for synchronization and must be taken simultaneously by both automata.

Definition 7 ([Alu99]) Consider two timed automata with disjoint sets of clocks $A_1 = \langle L_1, l_1^0, \Sigma_1, Cl_1, E_1, Inv_1 \rangle$ and $A_2 = \langle L_2, l_2^0, \Sigma_2, Cl_2, E_2, Inv_2 \rangle$. The *product automaton* $A_1 \| A_2$ is the timed automaton $\langle L_1 \times L_2, (l_1^0, l_2^0), \Sigma_1 \cup \Sigma_2, Cl_1 \cup Cl_2, Inv, E \rangle$, where $Inv(l_1, l_2) = Inv(l_1) \wedge Inv(l_2)$, and the edges are defined as follows:

1. For $a \in \Sigma_1 \cap \Sigma_2$, $\langle (l_1, l_2), a, g_1 \wedge g_2, r_1 \cup r_2, (l_1', l_2') \rangle \in E$ iff $\langle l_1, a, g_1, r_1, l_1' \rangle \in E_1$ and $\langle l_2, a, g_2, r_2, l_2' \rangle \in E_2$.

2. For $a \in \Sigma_1 \setminus \Sigma_2$, $\langle (l_1, l_2), a, g, r, (l_1', l_2) \rangle \in E$ iff $\langle l_1, a, g, r, l_1' \rangle \in E_1$ and $l_2 \in L_2$.

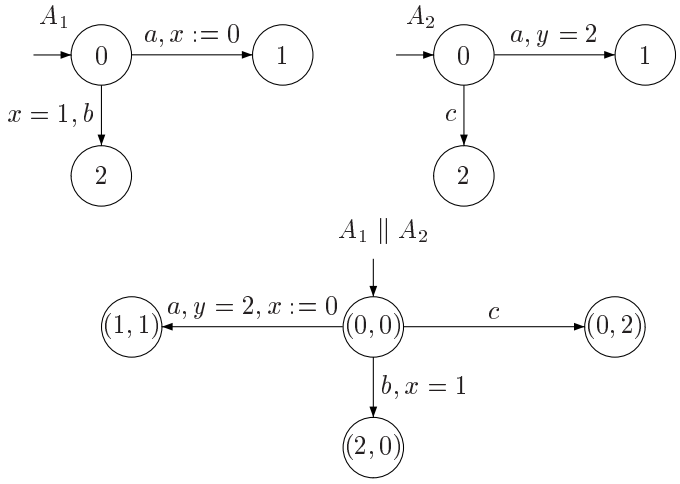

Figure 5.1: Product construction for timed automata.

3. For $a \in \Sigma_2 \setminus \Sigma_1$, $\langle (l_1, l_2), a, g, r, (l_1, l_2') \rangle \in E$ iff $\langle l_2, a, g, r, l_2' \rangle \in E_2$ and $l_1 \in L_1$.

Figure 5.1 illustrates two timed automata together with the resulting product automaton.

To encode the system $A_1 \| A_2$ into a $\mathcal{C}$-program, as described in Chapter 4 using Definition 2, the product automaton must be constructed first. For networks consisting of a large number of components, this leads to an exponential blowup in the number of resulting locations and transitions, and therefore also in the length of the Boolean constraint formulas. Here, we propose a method for encoding a network of timed automata into a $\mathcal{C}$-program in a compositional way, which does not require the construction of the product automaton.

For a timed automaton A with set of clocks Cl the formula $\mathit{fix}(A)$ is used to encode "inactivity", that is, the fact that A does not perform any transition.

$$\mathit{fix}(A) := (at' = at \ \wedge \bigwedge_{x \in Cl} x' = x).$$

Definition 8 Consider two timed automata $A_1 = \langle L_1, l_1^0, \Sigma_1, Cl_1, E_1, Inv_1 \rangle$ and $A_2 = \langle L_2, l_2^0, \Sigma_2, Cl_2, E_2, Inv_2 \rangle$. Further, let $\langle I_i, T_i \rangle$ be the program in $\mathsf{Prg}(\mathcal{C}(V_i \cup V_i'))$ corresponding to A_i, where $V_i = \{at_i, act_i\} \cup Cl_i$ and $V_i' = \{at_i'\} \cup Cl_i'$, and the variables act_i are interpreted over Σ_i, for $i = 1, 2$. The system $A_1 \| A_2$ can be encoded into a $\langle I, T \rangle$ program in $\mathsf{Prg}(\mathcal{C}(V \cup V'))$ over the set $V = V_1 \cup V_2$ and $V' = V_1' \cup V_2'$ in a compositional way, as follows.

- Initial state (l_1^0, l_2^0)

$$I^s := I_1 \wedge I_2 = (at_1 = l_1^0 \wedge at_2 = l_2^0 \wedge \bigwedge_{x \in Cl_1} x = 0 \wedge \bigwedge_{y \in Cl_2} y = 0)$$

- State transition step corresponding to $e_1 = \langle l_1, a, g_1, r_1, l_1' \rangle \in E_1$

$$\tilde{T}^s(e_1) = \begin{cases} at_1 = l_1 \wedge at_1' = l_1' \wedge g_1 \wedge \bigwedge_{x \in Cl_1} x' = z \wedge act_1 = a \wedge act_2 = a \\ \quad \text{iff } a \in \Sigma_1 \cup \Sigma_2 \\ at_1 = l_1 \wedge at_1' = l_1' \wedge g_1 \wedge \bigwedge_{x \in Cl_1} x' = z \wedge act_1 = a \wedge \mathit{fix}(A_2) \\ \quad \text{iff } a \in \Sigma_1 \setminus \Sigma_2 \end{cases}$$

where $z = 0$ if $x \in r_1$; otherwise $z = x$. The above formula is equal to

$$\tilde{T}^s(e_1) = \begin{cases} \tilde{T}_1(e_1) \wedge act_2 = a & \text{iff } \ a \in \Sigma_1 \cup \Sigma_2 \\ \tilde{T}_1(e_1) \wedge \mathit{fix}(A_2) & \text{iff } \ a \in \Sigma_1 \setminus \Sigma_2 \end{cases}$$

where $\tilde{T}_1(e_1)$ encodes e_1 independent of A_2, as illustrated in Definition 2.

- State transition step corresponding to $e_2 = \langle l_2, a, g_2, r_2, l_2' \rangle \in E_2$

$$\tilde{T}^s(e_2) = \begin{cases} \tilde{T}_2(e_2) \wedge act_1 = a & \text{iff } \ a \in \Sigma_1 \cup \Sigma_2 \\ \tilde{T}_2(e_2) \wedge \mathit{fix}(A_1) & \text{iff } \ a \in \Sigma_2 \setminus \Sigma_1 \end{cases}$$

- Delay steps

$$\begin{aligned} \mathit{delay}^s \quad := \quad \exists \delta \geq 0. \Big(& \bigwedge_{l \in Inv(A_1) \cup Inv(A_2)} (at = l \Rightarrow Inv(l)(Cl_1' \cup Cl_2')) \\ & \wedge \ (at_1' = at_1) \ \wedge \ (at_2' = at_2) \\ & \wedge \bigwedge_{x \in Cl_1} (x' = x + \delta) \ \wedge \bigwedge_{y \in Cl_2} (y' = y + \delta) \Big) \end{aligned}$$

where $at = at_1$ if $l \in Inv(A_1)$; otherwise $at = at_2$. The formula delay^s in quantifier-free form is equivalent to $\mathit{delay}_1 \wedge \mathit{delay}_2 \wedge (d_1 = d_2)$, where delay_1 and delay_2 describe the delay steps of A_1 respectively A_2 also in quantifier-free form. The conjunct $d_1 = d_2$ is used to relate the clock differences from delay_1 and delay_2. d_i denotes the clock difference $x_i' - x_i$ for $x_i \in Cl_i$ with $x_i' - x_i \geq 0$. Such a clock difference always exists in delay_i after quantifier elimination.

- Transition relation T^s

$$T^s := \Big(\big(\bigotimes_{e_1 \in E_1} \tilde{T}^s(e_1) \otimes \mathit{fix}(A_1) \big) \ \wedge \ \big(\bigotimes_{e_2 \in E_2} \tilde{T}^s(e_2) \otimes \mathit{fix}(A_2) \big) \Big) \bigotimes \mathit{delay}^s$$

The network consisting of the timed automata A_1 and A_2 from Figure 5.1, for example, is defined as a program over the set of variables

$$V = \{at_1, at_2, act_1, act_2, at_1', at_2', x, y, x', y'\},$$

where $\mathit{fix}(A_1) = (at_1' = at_1 \wedge x' = x)$ and $\mathit{fix}(A_2) = (at_2' = at_2 \wedge y' = y)$.

$$\begin{array}{lcl}
I^s & := & (at_1 = 0 \wedge at_2 = 0 \wedge x = 0 \wedge y = 0) \\
T^s & := & \Big[[(at_1 = 0 \wedge at_1' = 1 \wedge x' = 0 \wedge act_1 = a \wedge act_2 = a) \otimes \\
 & & \quad (at_1 = 0 \wedge at_1' = 2 \wedge x = 1 \wedge x' = x \wedge act_1 = b \wedge \mathit{fix}(A_2)) \otimes \ \mathit{fix}(A_1)] \ \wedge \\
 & & \quad [(at_2 = 0 \wedge at_2' = 1 \wedge y = 2 \wedge y' = y \wedge act_2 = a \wedge act_1 = a) \otimes \\
 & & \quad (at_2 = 0 \wedge at_2' = 2 \wedge y' = y \wedge act_2 = c \wedge \mathit{fix}(A_1)) \otimes \ \mathit{fix}(A_2)]\Big] \otimes \\
 & & (at_1' = at_1 \wedge at_2' = at_2 \wedge x' - x \geq 0 \wedge y' - y = x' - x)
\end{array}$$

Theorem 4 (BMC for Networks of Timed Automata) Given two timed automata with disjoint set of clocks $A_i = \langle L_i, l_i^0, \Sigma_i, Cl_i, E_i, Inv_i \rangle$, for $i = 1, 2$. Let $M^s = \langle I^s, T^s \rangle$ be the program corresponding to the network $A_1 \| A_2$ as given in Definition 8, and $M = \langle I, T \rangle$ be the program encoding the product automaton $A_1 \times A_2$ according to Definition 2. Then for a $k \in I\!N$, the kth unfoldings of M^s and M are equisatisfiable, that is $[\![M^s]\!]_k \equiv [\![M]\!]_k$.

Proof. ($\Leftarrow$) Assume $[\![M]\!]_k = I_0(x[0]) \wedge \bigwedge_{j=0}^{k-1} T_j(x[j], x[j+1])$ as given in Definition 4. Let us first consider only state transition steps for M. We prove by induction over k that if $[\![M]\!]_k$ is satisfiable then so $[\![M^s]\!]_k$.

Basis case k=0. By Definition 7 (product construction)

$$[\![M]\!]_0 = I_0(x[0]) = (at[0] = (l_1^0, l_2^0) \ \wedge \bigwedge_{x_i \in Cl_1} x_i[0] = 0 \ \wedge \bigwedge_{y_i \in Cl_2} y_i[0] = 0)$$

This formula can be transformed into an equisatisfiable formula of the form

$$(at_1[0] = l_1^0 \ \wedge \ at_2[0] = l_2^0 \ \wedge \bigwedge_{x_i \in Cl_1} x_i[0] = 0 \ \wedge \bigwedge_{y_i \in Cl_2} y_i[0] = 0)$$

which by Definition 8 equals I^s. Thus, $I_0(x[0])$ and $I_0^s(x[0])$ are equisatisfiable.

Induction hypothesis. For $j < k$, $[\![M^s]\!]_j$ is satisfiable if $[\![M]\!]_j$ is satisfiable.

Induction step j=k. We show that $\bigwedge_{j=0}^{k} T_j^s$ is satisfiable if $\bigwedge_{i=0}^{k} T_i$ is satisfiable. By induction hypothesis it follows that $\bigwedge_{j=0}^{k-1} T_j^s$ and $\bigwedge_{i=0}^{k-1} T_i$ are equisatisfiable. To have a closer look at T_k, we consider a state transition corresponding to an edge $e = \langle (l_1, l_2), a, g, r, (l_1', l_2') \rangle \in E$. We distinguish three

cases: A_1 and A_2 synchronize on a, only A_1 (A_2) performs a, or the transition from (l_1, l_2) to (l_1', l_2') is not allowed. In the first case e is obtained from $e_1 = \langle l_1, a, g_1, r_1, l_1', \rangle \in E_2$ and $e_2 = \langle l_2, a, g_2, r_2, l_2', \rangle \in E_2$, and we have

$$\begin{aligned}\tilde{T}_k \quad := \quad & (at[k] = (l_1, l_2) \ \wedge \ act[k] = a \ \wedge \ g \ \wedge \ at[k+1] = (l_1', l_2') \ \wedge \\ & \bigwedge_{x_i \in Cl_1} x_i[k+1] = z_i \ \wedge \ \bigwedge_{y_i \in Cl_2} y_i[k+1] = z_i)\end{aligned}$$

where $g = g_1 \wedge g_2$, and $z_i = 0$ if $x_i \in r_1 \cup r_2$; otherwise $z_i = x_i[k]$. From $at[k] = (l_1, l_2)$ $(at[k+1] = (l_1', l_2'), act[k] = a)$ satisfiable it follows $at_1[k] = l_1 \wedge at_2[k] = l_2$ $(at_1[k+1] = l_1' \wedge at_2[k+1] = l_2', act_1[k] = a \wedge act_2[k] = a)$ satisfiable. Therefore, the formulas $\tilde{T}^s(e_1)$ and $\tilde{T}^s(e_2)$ are both satisfiable in step k, and by Definition 8 the formula T_k^s is satisfiable. In the second case $a \in \Sigma_1 \setminus \Sigma_2$, $e_1 = \langle l_1, a, g_1, r_1, l_1', \rangle$, and

$$\begin{aligned}\tilde{T}_k \quad := \quad & (at[k] = (l_1, l_2) \ \wedge \ act[k] = a \ \wedge \ g \ \wedge \ at[k+1] = (l_1', l_2) \ \wedge \\ & \bigwedge_{x_i \in Cl_1} x_i[k+1] = z_i \ \wedge \ \bigwedge_{y_i \in Cl_2} y_i[k+1] = z_i)\end{aligned}$$

where $g = g_1$, and $z_i = 0$ if $x_i \in r_1$; otherwise $z_i = x_i[k]$. By an argument similar to that of the first case, $\tilde{T}^s(e_1)$ is satisfiable in step k. Since A_2 does not perform any transition, $\mathit{fix}(A_2)$ is satisfiable, and therefore T_k^s are satisfiable. In the third case, if the transition e cannot be taken, the formulas $\mathit{fix}(A_1)$ and $\mathit{fix}(A_2)$, and therefore T^s, are satisfiable.

($\Rightarrow$) Follows by a similar argumentation.

Now, let us consider delay steps. According to Definition 2, the delay steps of the system $A_1 \times A_2$ (after quantifier elimination) are encoded as

$$\begin{aligned}\mathit{delay} \quad := \quad & \bigwedge_{l \in Inv(A_1 \times A_2)} (at = l \Rightarrow Inv(l)(Cl_1' \cup Cl_2')) \\ & \wedge \ at' = at \ \wedge \ x_1' - x_1 \geq 0 \ \wedge \ \bigwedge_{y \in Cl_1 \cup Cl_2 \setminus \{x_1\}} y' - y = x_1' - x_1\end{aligned}$$

The above formula is equivalent to

$$\begin{aligned}\mathit{delay} \quad := \quad & \bigwedge_{l \in Inv(A_1)} (at_1 = l \Rightarrow Inv(l)(Cl_1')) \ \wedge \\ & \bigwedge_{l \in Inv(A_2)} (at_2 = l \Rightarrow Inv(l)(Cl_2')) \ \wedge\end{aligned}$$

$$
\begin{aligned}
& at_1' = at_1 \ \wedge \ at_2' = at_2 \ \wedge \\
& x_1' - x_1 \geq 0 \ \wedge \ x_2' - x_2 \geq 0 \ \wedge \\
& \bigwedge_{y \in Cl_1 \setminus \{x_1\}} y' - y = x_1' - x_1 \ \wedge \\
& \bigwedge_{y \in Cl_2} y' - y = x_1' - x_1
\end{aligned}
$$

which is equal to $delay_1 \ \wedge \ delay_2 \ \wedge \ (x_1' - x_1 = x_2' - x_2)$. Thus, by Definition 8, $delay = delay^s$.

□

Chapter 6

Discussion and Conclusion

We presented a bounded model checking procedure (BMC) for timed automata and linear temporal logic with real-valued clock constraints. The main contribution is a complete BMC algorithm for timed automata, which is compositional in that Boolean constraint formulas encoding complex systems can be obtained by Boolean combinations of the encoding of the components. A direct encoding of the product automaton would cause an exponential blow up in the length of the resulting Boolean constraint formula. The completeness proof can be adapted to any systems with a finite bisimulation. Further, we give lower and upper bounds for the length k of counterexamples, that depend on the structure of the Büchi automaton of the given formula, and the region automaton corresponding to the timed automaton.

The main problem of the BMC approach is to come up with efficient algorithms for solving the satisfiability problem for Boolean constraint formulas. Specialized data structures for timed automata, such as difference bounded matrices (DBMs) [Dil89], clock difference diagrams (CDDs) [LPWY99], or difference decision diagrams (DDDs) [MLAH99], cannot be applied directly for BMC, since the generated formulas contain clock constraints of the form $x' - x = y' - y$, as needed for encoding the delay steps. It is unclear if even recently developed constraint solvers, such as the satisfiability checker for difference logic, presented in [MNAM02], can deal with this kind of constraints. On the other hand, general-purpose theorem proving, such as PVS [ORS92], which uses a combination of BDDs [Bry86] and linear arithmetic reasoning based on loop

residue [Sho81], is not very efficient. For example, finding a counterexample of length $k = 2$ in the (modified) train gate controller protocol requires around 70 s, and for $k = 3$ around 8500 s. Recently, new techniques for checking satisfiability of Boolean constraint formulas have been developed, by combining SAT solvers with domain-specific decision procedures based on lemmas on demand [dMRS02, BDS02]. We have implemented a prototypical satisfiability solver [dMRS02] that combines the SAT solver Chaff [MMZ+01] with the decision procedures ICS [FORS01]. The core of the solver is a refinement algorithm based on lazy theorem proving. In each refinement step, the Boolean satisfiability checker Chaff is used to suggest candidate assignments. Then ICS checks whether such a Boolean assignment determines a consistent assignment for the corresponding set of constraints. Whenever such a consistency check fails, the current Boolean formula is refined by adding a Boolean analogue of this inconsistency. The SAT solver is restarted, and a new candidate assignment for the refined formula is suggested.

We have performed some initial experiments, using Fischer's mutual exclusion protocol [Lam87] with a slight modification of the timing constraints as a benchmark. We encoded systems of $n = 2, \ldots, 10$ processes as a Boolean constraint formula in a compositional way, as described in Chapter 5. On a Pentium II, 450 MHz, for 2 processes we found a counterexample of length 3 (shortest counterexample) in 0.23 s, of length 5 in 0.85 s, and of length 10 in 6.12 s. For 5 processes we obtain, for $k = 5$, 1.34 s, and for $k = 10$, 16.11 s. For a system consisting of 10 processes, and a bound $k = 10$ a counterexample was found in 210.8 s. Although in an initial phase, the performed experiments show that BMC is a promising technique for verifying timed systems. Errors in larger systems for which conventional timed model checking tools fail or are inefficient can be found using BMC.

Lazy theorem proving and lemmas on demands are relatively new concepts, and the underlying implementations improve on a daily basis. Currently, we perform our experiments also with both a new implementation of lemmas on demand and CVC [BDS00], and are in the process of evaluating and comparing both approaches.

Bibliography

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. *5th Symp. on Logic in Computer Science (LICS 90)*, pages 414–425, 1990.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 25 April 1994.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford University, 1991.

[Alu99] R. Alur. Timed automata. *Lecture Notes in Computer Science*, 1633:8–22, 1999.

[BDS00] Clark W. Barrett, David L. Dill, and Aaron Stump. A framework for cooperating decision procedures. In *17th International Conference on Computer-Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–97. Springer-Verlag, 2000.

[BDS02] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT, 2002. To be presented at CAV 2002.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[CBRZ01] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CFF+01] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking in an industrial setting. In *Computer-Aided Verification, CAV 2001*, volume 2101 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, July 2001.

[Dam96] Dennis René Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, July 1996.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212, Grenoble, France, 1989. Springer-Verlag.

[dMRS02] Leonardo de Moura, Harald Rueß, and Maria Sorea. Lazy theorem proving for bounded model checking over infinite domains, 2002. Accepted for publication at CADE'02.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. *Lecture Notes in Computer Science*, 1066:208–219, 1996.

[FORS01] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated canonizer and solver. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV'2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer-Verlag, 2001.

[GPVW95] Rob Gerth, Doron Peled, Moshe Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *Lecture Notes in Computer Science*, 1254:460–463, 1997.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, June 1994.

[Lam87] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, February 1987.

[LPWY99] Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, Fall 1999.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[MLAH99] J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Difference decision diagrams. In *Computer Science Logic*, The IT University of Copenhagen, Denmark, September 1999.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.

[MNAM02] Moez Mahfoudh, Peter Niebert, Eugene Asarin, and Oded Maler. A satisfiability checker for difference logic, 2002. To be presented at the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 02), Ohio.

[MRS02] M. Oliver Möller, Harald Rueß, and Maria Sorea. Predicate abstraction for dense real-time systems. *Electronic Notes in Theoretical Computer Science*, 65(6), 2002.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, 1992.

[Sho81] Robert Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, October 1981.

[Sor01] Maria Sorea. Tempo: A model-checker for event-recording automata. *Workshop on Real-Time Tools*, Aalborg, Denmark, August 2001. Full version available as Technical Report SRI-CSL-01-04, Computer Science Laboratory, SRI International, Menlo Park, CA, 2001.

[TY01] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, January 2001. Kluwer Academic Publishers.