Automatic Verification of Real-Time Communicating Systems by Constraint-Solving

Wang Yi, Paul Pettersson, and Mats Daniels

Department of Computer Systems, Uppsala University. Sweden.
Email: {yi,paupet,matsd}@docs.uu.se.

Abstract. In this paper, an algebra of timed processes with real-valued clocks is presented, which may serve as a description language for networks of timed automata. We show that requirements such as "a process will never reach an undesired state" can be verified by solving a simple class of constraints on the clock-variables. A symbolic on-the-fly reachability algorithm for the language has been developed and implemented as a software tool based on constraint-solving techniques. To our knowledge, this is the first on-the-fly verification algorithm for timed automata. In fact, the tool is the very first implementation of the UPPAAL tool.
As examples, we model and verify safety properties of a real-time mutual exclusion protocol and a railway crossing controller.

1 Introduction

During the past few years, various formal techniques for modeling and verifying real-time systems have been put forwards, e.g. automata based [ACD90,ACH+92,AD90,HNSY92] and process algebra based [Cer92,CGL93,HLY92,MT91,Yi91]. One of the most successful approaches is the *timed automata* model due to Alur and Dill [ACD90], which is the classical finite-state automaton model extended with clock variables modeling time delays.

In this paper, we study real-time communicating systems. Such a system consists of a number of components with their own or shared clocks. The components may communicate with each other and the environment through channels according to the timing constraints on the values of the clocks. Naturally, we can use timed automata to describe the components. However, it is not obvious how to combine the component descriptions in a description of the whole system. Traditionally, the parallel composition of timed automata is interpreted as logical conjunction, which is similar to the strong (multi-) synchronisation operator from process algebras, defined by the rule:

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q'}{P \,\&\, Q \xrightarrow{a} P' \,\&\, Q'}$$

Intuitively, it means that the whole system described by $P \,\&\, Q$ may make a move (i.e. do an a) only if the components described by P and Q can do the same. That is, all components of a concurrent system must synchronise on every action at every time point. This seems to be a strong restriction for practical applications of timed automata. Real systems are often highly distributed and in many cases, a system component may only want to communicate with the environment or a particular component, without synchronising with the others. Therefore, we introduce a CCS-like parallel composition operator [Mil89] for timed automata, to describe one-to-one communication and interleaving.

As the first contribution of this paper, we present an algebra for networks of timed automata, which provides a number of algebraic operators including the parallel composition operator to model communication and concurrency. The operators can be used to construct complex automata (i.e. complex system descriptions) in terms of simpler ones (i.e. component descriptions). Thus, the algebra may serve as a *structural* description language for real-time communicating systems.

As the second contribution, we develop an on-the-fly verification algorithm based on constraint-solving techniques, for the type of systems described above. There have been a number of verification techniques developed for timed automata, e.g. [ACH+92,ACD90,HNSY92]. However, these

algorithms always construct the product of the automata before checking properties of the system. Even for moderately sized systems, explicit representation and exploration of the product automaton is infeasible as it grows exponentially with both the number of components and the number of clocks in the system. Though there have been efficient minimisation techniques, such as [ACH+92], the problem of state-space explosion is still an obstacle for automatic verification.

It has been a well recognised fact (e.g. [Hal93]) that the practical goal of verification of real-time systems is often to verify simple logical properties, which do not need the whole power of model-checking (e.g. for timed CTL). We shall only consider simple safety-properties, which can be verified by checking if a certain set of states of the system is reachable or not. For instance, a railway control system (see Section 6) should guarantee that "at most one train can cross a critical point at the same time". This is a typical safety-property meaning that bad things can never happen. However, we can also verify properties requiring that a good thing will eventually happen within a certain time limit. For example, "a train should be able to pass a critical point (such as a bridge), within a bounded delay".

We shall present a symbolic on-the-fly verification algorithm for networks of timed automata. It is *symbolic* in the sense that the inherently infinite state-space of timed automata is finitely partitioned into subsets which are represented and manipulated using a class of linear constraints, known as *difference bound matrices* [Bel57,Dil89]. This allows the partitioning to take into account both the automata and the logical property currently being checked, to make the partitioning as coarse (and small) as possible. The algorithms operates in an *on-the-fly* manner [Hol91] in the sense that the checking of a particular property and the construction of the reachable part of the (symbolic) state-space is performed simultaneously. This avoids generating unnecessary parts of the state-space as the algorithm is stopped when the truth-hood of the currently checked property is determined, in contrast to the traditional approach of constructing the full state-space of the system before the checking.

The rest of the paper is organised as follows: In section 2, we present an algebra of timed processes, in which a syntactical term describes a network of timed automata; any timed automaton can be expressed in the algebra. Section 3 and 4 describe two symbolic semantics of processes, of which the latter is shown to yield a finite number of symbolic states. In section 5, we study the reachability problem associated with the algebra. An algorithm based on the finite symbolic semantics is presented, and proved to be sound and complete. In section 6, as examples, we study a simple railway control system and variant of Fischer's mutual exclusion protocol. Section 7 concludes the paper.

2 An Algebra of Processes with Clocks

Process algebras provide a clean and general paradigm for compositional specification of communicating processes. In this section we present an algebra of timed automata, serving as a structural specification language for real-time communicating systems. The idea is to use algebraic operators to construct complex system specifications in terms of simpler ones (or component specification).

2.1 Syntax

Traditionally, a prefix expression $\alpha.P$ in process algebras describes a process which may perform an α-transition and then continue with P. But no timing information is given on when the transition may be taken.

Following Alur and Dill [AD90], we assume a set of clocks to specify timing constraints on transitions. Conceptually, the clocks may be considered as the system clocks of a concurrent system, owned or shared by processes in the system. The processes may test the clocks by comparing the clock values with integer constants and reset the clocks (i.e. assigning clock values to 0). Further, assume that all clocks proceed at the same rate and measure the amount of time that has been elapsed since they were reset or started.

We extend the action prefix $\alpha.P$ to the form $(g, \alpha, r).P$ where g is a predicate over the clock values and r is a subset of clocks to be reset. Intuitively, $(g, \alpha, r).P$ describes a timed process which may perform an α-transition instantaneously when g is true of the current clock values and then continue as P with the clocks in r being reset (and the other clocks proceeding with their old values).

We also introduce a way to force processes to make progress. We use an invariant operator in the form $g \rhd\!\gg P$, where g is a predicate over clock values [HNSY92,DB96]. Intuitively, the process may idle or go on as P while g is satisfied by the system clocks.

Definition 1 (Clock Constraints). *Let $\mathcal{C}$ denote a set of clocks, ranged over by x, y ,z. We use $\mathcal{B}(\mathcal{C})$ to stand for the set of logical formulae generated by the following syntax:*

$$g ::= x \sim n \mid x - y \sim n \mid g \wedge g$$

where $\sim \in \{<, >, =, \leq, \geq\}$ and n is a natural number. □

We shall use $\mathtt{tt}$ to denote a clock constraint like $x \geq 0$ that is always satisfied, and $\mathtt{ff}$ for $x < 0$ that is always false as clocks can only have non-negative values. Note that we could allow a more general form of formulas such as disjunction $g \vee g$. However, it will not give more expressive power to the description language we are going to develop. In fact, logical disjunction can be modeled by the behavioural choice operator.

The language is essentially CCS [Mil89] extended with the timed action prefix $(g, \alpha, r).P$ and the invariant operator $g \rhd\!\gg P$. Let $\mathcal{A}$ be a finite set of actions ranged over by α, β etc. We use $\mathcal{L} = \{\alpha \mid \alpha \in \mathcal{A}\} \cup \{\bar{\alpha} \mid \alpha \in \mathcal{A}\}$[1] with $\bar{\bar{\alpha}} = \alpha$ for representing external actions, a distinct symbol τ representing internal actions, and $\mathcal{A}ct = \mathcal{L} \cup \{\tau\}$ ranged over by a, b for representing both internal and external actions. Further, assume a set of process variables ranged over by X, Y (and sequences of letters).

We shall see that the algebraic structure of a process expression P represents the control-structure of a process. This will be clear when we present the operational semantics. We adopt a two-phase syntax according to two types of control-structures: *regular* and *concurrent*.

We start with processes whose control-structure are regular in the sense that no concurrency is involved. The regular process expressions are generated by the following grammar:

$$E ::= \mathsf{nil} \mid (g, a, r).E \mid g \rhd\!\gg E \mid E + F \mid X \mid X \stackrel{def}{=} E$$

We shall restrict expressions to be *well-guarded* in the following sense:

Definition 2 (Well-Guarded Expressions). *X is well-guarded in E if and only if every free occurrence of X in E is within a subexpression (a guard) of the form $(g, a, r).F$. E is well-guarded if and only if every free variable in E is well-guarded in E, and for every subexpression of the form $X \stackrel{def}{=} F$ in E, X is well-guarded in F.* □

Let $\mathcal{P}$ denote the set of well-guarded expressions generated by the grammar above. We call $\mathcal{P}$ *regular timed processes.*

We shall study concurrent processes in the form $(P_1 \mid ... \mid P_n)\backslash L$, where $P_i \in \mathcal{P}$ describing the components and $L \subseteq \mathcal{L}$ representing the set of internal channels connecting the components. We use $\mathcal{P}$ to denote the set of timed concurrent processes, ranged over by P, Q and R.

For simplicity, we have ignored the relabelling operator. The results of this paper can easily be extended to more general types of processes modeled by the combination of parallel composition, restriction and relabelling.

[1] The action $\bar{\alpha}$ is called the co-action of α. In our examples, we shall use $\alpha!$ instead of $\bar{\alpha}$ to denote an *output* event and $\alpha?$ instead of α to denote an *input* event.

$$\frac{g(u)}{((g,\alpha,r).E,u) \overset{a}{\rightsquigarrow} (E,r[u])} \qquad \frac{(E,u) \overset{a}{\rightsquigarrow} (E,u')}{(g \rhd\!\gg E,u) \overset{a}{\rightsquigarrow} (E,u')}$$

$$\frac{(F,u) \overset{a}{\rightsquigarrow} (F',r[u])}{(E+F,u) \overset{a}{\rightsquigarrow} (F',r[u])} \qquad \frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(E+F,u) \overset{a}{\rightsquigarrow} (E',r[u])}$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(X,u) \overset{a}{\rightsquigarrow} (E',r[u])} \quad [X \overset{def}{=} E]$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(E|F,u) \overset{a}{\rightsquigarrow} (E'|F,r[u])} \qquad \frac{(F,u) \overset{a}{\rightsquigarrow} (F',r[u])}{(E|F,u) \overset{a}{\rightsquigarrow} (E|F',r[u])}$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u]) \quad (F,u) \overset{\bar{a}}{\rightsquigarrow} (F',q[u])}{(E|F,u) \overset{\tau}{\rightsquigarrow} (E'|F',(r \cup q)[u])}$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(E\backslash L,u) \overset{a}{\rightsquigarrow} (E'\backslash L,r[u])} \quad [a,\bar{a} \notin L]$$

Table 1. The Action Transition Relation.

2.2 Operational Semantics

We interpret $\mathcal{P}$ using clock assignments. Let $\mathbb{R}_+$ stand for the non-negative real numbers. A *clock assignment* $u : \mathcal{C} \mapsto \mathbb{R}_+$ is a function mapping each clock x to a non-negative real $u(x)$. Assume that $d \in \mathbb{R}_+$ and $r \subseteq \mathcal{C}$ is a set of clocks. We use $u \oplus d$ to denote the clock assignment which maps each clock x to $u(x)+d$, and $r[u]$ to denote the clock assignment which maps x to 0 if $x \in r$ and to $u(x)$ otherwise. Furthermore, given a clock constraint $g \in \mathcal{B}(\mathcal{C})$, we write $g(u)$ to mean the truth value of g, relative to assignment u.

To interpret $\mathcal{P}$ we also define for each control-node $E \in \mathcal{P}$ an *invariant condition*, denoted $I(E)$. Intuitively, $I(E)$ restricts the amount of time the process may idle in E, i.e. it must switch to another control-node while $I(E)$ still holds. It is defined as follows:

Definition 3 (Invariant Condition). *Assume $E, F \in \mathcal{P}$. Let $I(E)$ be the invariant condition of E, defined inductively as:*

$$\begin{aligned} I(\mathsf{nil}) &= \mathsf{tt} \\ I((g,a,r).E) &= \mathsf{tt} \\ I(g \rhd\!\gg E) &= g \wedge I(E) \\ I(E+F) &= I(E) \wedge I(F) \\ I(X) &= I(E) \text{ if } X \overset{def}{=} E \\ I(E|F) &= I(E) \wedge I(F) \\ I(E\backslash L) &= I(E) \end{aligned}$$

□

A *state* of a process is a pair (P,u) where $P \in \mathcal{P}$ stands for the current control-node and u denotes the current clock values. A process may make two types of transitions from state to state:

Definition 4 (Transition Rules). *Assume $a \in \mathcal{A}ct$ and $d \in \mathbb{R}_+$:*

- *(Action) $(P,u) \overset{a}{\rightsquigarrow} (P',u')$ following the rules defined in Table 1.*
- *(Delay) $(P,u) \overset{d}{\rightsquigarrow} (P,u \oplus d)$ if $I(E)(u)$ and $I(E)(u \oplus d)$.*

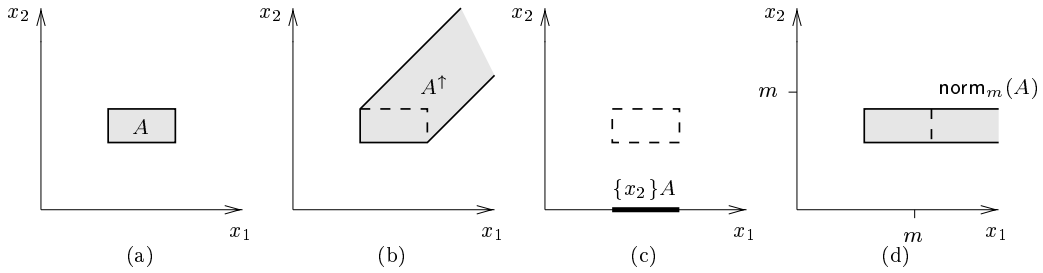

Fig. 1. Operations on Clock Constraints.

We will use $(P^0, u^0) \rightsquigarrow^n (P^n, u^n)$ *to denote* $(P^0, u^0) \overset{\sigma_1}{\rightsquigarrow} (P^1, u^1) \overset{\sigma_2}{\rightsquigarrow} \ldots \overset{\sigma_{n-1}}{\rightsquigarrow} (P^{n-1}, u^{n-1}) \overset{\sigma_n}{\rightsquigarrow} (P^n, u^n)$ *for* $\sigma_i \in \mathcal{A}ct \cup \mathbb{R}_+$, *and* $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ *if* $(P^0, u^0) \rightsquigarrow^n (P^f, u^f)$ *for some finite number* n. □

The delay transition relation describes the pure passing of time; the action transition relation describes the instantaneous occurrence of actions and, possibly, the resetting of clocks.

Example 1. Consider the algebraic terms X, P and Q:

$$\begin{aligned} X &\overset{def}{=} (y \leq 1) \rhd\!\!> (\mathtt{tt}, \tau, \{\}).P \\ P &\overset{def}{=} (x \leq 2) \rhd\!\!> \Big((x < 1, \tau, \{\}).Q + (x \geq 1, a, \{x\}).P \Big) \\ Q &\overset{def}{=} \mathsf{nil} \end{aligned}$$

The control-node X will become P by doing a τ-action before the clock value of y becomes greater than 1. If the value of y is greater than 1, X is deadlocked. However, when the value of x is greater than, or equal to 1, the control-node of P will remain the same, i.e. P, but the τ-action will be disabled and the a-action will be enabled while x is less than, or equal to 2. For instance, X can perform the following sequence of transitions:

$$(X, 0, 0) \overset{\tau}{\rightsquigarrow} (P, 0, 0) \overset{m}{\rightsquigarrow} (P, m, m) \overset{a}{\rightsquigarrow} (P, 0, m) \overset{\tau}{\rightsquigarrow} (Q, 0, m)$$

for all $m \in [1, 2]$.

3 Symbolic Semantics of Processes

Clearly, the concrete semantics of processes defined in the previous section yields an infinite transition system due to the real-valued clocks. In this section we shall give a symbolic semantics of processes which will be used in the next section to develop a finite partitioning of the state-space.

We consider symbolic states that are sets of concrete states sharing the same control-node. To represent sets of clock assignments we use clock constraints $\mathcal{B}(\mathcal{C})$ introduced in Section 2.1. Let D range over $\mathcal{B}(\mathcal{C})$. The key idea is to let D represent the set of clock assignments that are its solutions, and use states in the form (P, D) to *symbolically* represent the set of all states (P, u) such that u satisfies D.

3.1 Operations on Clock Constraints

We will need a few *operations* on clock constraints to define the symbolic semantics. Given a clock constraint D we call the set of clock assignments satisfying D, the *solution set* of D.

$$\frac{}{(g,a,r).E \xrightarrow{g\,a\,r} E} \qquad \frac{E \xrightarrow{g\,a\,r} E'}{\psi \rhd\!\!> E \xrightarrow{g\,a\,r} E'}$$

$$\frac{E \xrightarrow{g\,a\,r} E'}{E + F \xrightarrow{g\,a\,r} E'} \qquad \frac{F \xrightarrow{g\,a\,r} F'}{E + F \xrightarrow{g\,a\,r} F'}$$

$$\frac{E \xrightarrow{g\,a\,r} E'}{X \xrightarrow{g\,a\,r} E'} \quad [\ X \stackrel{def}{=} E\ ]$$

$$\frac{E \xrightarrow{g\,a\,r} E'}{E|F \xrightarrow{g\,a\,r} E'|F} \qquad \frac{F \xrightarrow{g\,a\,r} F'}{E|F \xrightarrow{g\,a\,r} E|F'}$$

$$\frac{E \xrightarrow{g\,a\,r} E' \quad F \xrightarrow{f\,\bar{a}\,s} F'}{E|F \xrightarrow{h\,\tau\,q} E'|F'} \begin{bmatrix} h = g \wedge f \\ q = r \cup s \end{bmatrix}$$

$$\frac{E \xrightarrow{g\,a\,r} E'}{E\backslash L \xrightarrow{g\,a\,r} E'\backslash L} \quad [\ a, \bar{a} \notin L\ ]$$

Table 2. Transition Rules for Control-Nodes.

Definition 5. *Let A and A' be solution sets of clock constraint D and D'. We define:*

$$\begin{aligned} A \wedge A' &= \{\ u \mid u \in A \text{ and } u \in A'\ \} \\ A^{\uparrow} &= \{\ u \oplus d \mid d \in \mathbb{R}_+ \text{ and } u \in A\} \\ \{x\}A &= \{\ x[u] \mid u \in A\ \} \end{aligned}$$

□

First note that $A \wedge A'$ is simply the intersection of the two sets. Consider the set A for the case of two clocks, shown in Figure 1(a). The two operations $A^{\uparrow}$ and $\{x\}A$ are illustrated in (b) and (c) of Figure 1 respectively[2]. Intuitively, $A^{\uparrow}$ is the largest set of clock assignments that will eventually be reached from A after some delay, and the reset-operation $\{x_2\}A$ is the projection of A down on the x_1-axis. We extend the reset-operation $\{x\}A$ to sets of variables. Assume that r is a set of clock variables and $r = \{x\} \cup r'$. We define $r(A) = \{x\}(\{r'\}A)$ and $\{\}A = A$. In order to save notation, from now on we shall simply use $D \wedge D'$, $D^{\uparrow}$ and $\{x\}D$ to denote the solution sets $A \wedge A$, $A^{\uparrow}$ and $\{x\}A$.

We also need two *predicates* over clock constraints in the semantics. We write $D \subseteq D'$ to mean that the solution set of D is included in the solution set of D' and $D = \emptyset$ to mean that the solution set of D is empty (i.e. D is not satisfiable).

3.2 Symbolic Transition Rules

In this section we will define the transition rules for symbolic states. First, we need to study the control-structure of processes more carefully.

We will interpret algebraic terms $\mathcal{P}$ as timed automata with location invariants[3] [AD90,HNSY92]. It should be obvious that each term $P \in \mathcal{P}$ describes a timed automaton with location invariants, i.e. $\langle \mathcal{P}, P, \longrightarrow, I \rangle$, where $\mathcal{P}$ (the set of algebraic terms) is the set of nodes, $P \in \mathcal{P}$ the initial node, $\longrightarrow$ is the least transition relation defined by the rules in Table 2, and $I : \mathcal{P} \mapsto \mathcal{B}(\mathcal{C})$ is the invariant

[2] Figure 1(d) is to illustrate the normalisation operator that will be introduced in the next section.

[3] Timed automata with location invariants are called "timed safety automata" in [HNSY92].

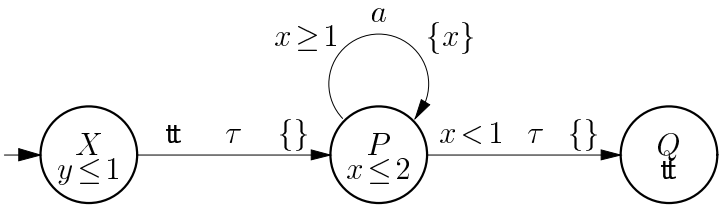

Fig. 2. Timed Automaton A_X.

assignments function as defined in Definition 3. In particular, note that $\mathcal{P}$ is finite. Note also that the terms $P \in \mathcal{P}$ are in the form $P = (P_1 | \ldots | P_n) \setminus L$, where $P_i \in \mathcal{P}$ and $L \subseteq \mathcal{L}$. Thus, rather than describing a single timed automaton, an algebraic term e.g. P may describe a whole *network of timed automata* P_i that communicate by synchronising pairwise on the internal channels (in L) connecting them.

Example 2. Reconsider the algebraic terms X, P and Q of Example 1. The timed automaton A_X described by X is shown in Figure 2. It has three control-nodes, X, P and Q, two clocks x and y, and three edges. The edge between P and Q has $x < 1$ as guard, τ as action and empty reset set $\{\}$. The invariant condition of the control-node P is $x \leq 2$.

We are now ready to define the symbolic semantics of processes. It is given by a transition system with the set of states being the *symbolic states* (P, D) and the transition relation defined by the following two rules:

Definition 6 (Symbolic Transition Rules). *Assume $a \in \mathcal{A}ct$ and a new symbol ε representing delays.*

- *(Action)* $(P, D) \overset{a}{\longmapsto} (P', r(D \wedge g))$ *if* $P \overset{g\,a\,r}{\longrightarrow} P'$.
- *(Delay)* $(P, D) \overset{\varepsilon}{\longmapsto} (P, D^{\uparrow P})$,

where $D^{\uparrow P} = (D \wedge I(P))^{\uparrow} \wedge I(P)$, and $I(P)$ is the invariant condition of P.

We will use $(P^0, D^0) \longmapsto^n (P^n, D^n)$ to denote an alternating sequence in the form $(P^0, D^0) \overset{\varepsilon}{\longmapsto} (P^1, D^1) \overset{a_2}{\longmapsto} (P^2, D^2) \overset{\varepsilon}{\longmapsto} \ldots \overset{a_{n-1}}{\longmapsto} (P^{n-1}, D^{n-1}) \overset{\varepsilon}{\longmapsto} (P^n, D^n)$ for $a_i \in \mathcal{A}ct$, and $(P^0, D^0) \longmapsto^\star (P^f, D^f)$ if $(P^0, D^0) \longmapsto^n (P^n, D^n)$ for some n. □

Intuitively, in the action transition relation $(P', r(D \wedge g))$ is the strongest post-condition of (P, D) after a transition $P \overset{g\,a\,r}{\longrightarrow} P'$. In the delay transition relation, $D^{\uparrow P}$ is the largest set of clock assignments that can be reached from D by delaying in P while the invariant condition $I(P)$ still holds.

Example 3. Consider the timed automaton of Figure 2. It has the following typical symbolic transition sequence:

$$
\begin{aligned}
&(X, (x = y = 0)) \overset{\varepsilon}{\longmapsto} (X, (x = y \wedge y \leq 1)) \overset{\tau}{\longmapsto} \\
&(P, (x = y \wedge y \leq 1)) \overset{\varepsilon}{\longmapsto} (P, (x = y \wedge y \leq 2)) \overset{a}{\longmapsto} \\
&(P, (x = 0 \wedge 1 \leq y \leq 2)) \overset{\varepsilon}{\longmapsto} \\
&(P, (x \leq 2 \wedge 1 \leq y \leq 4 \wedge 1 \leq y - x \leq 2)) \overset{\tau}{\longmapsto} \\
&(Q, (x < 1 \wedge 1 \leq y - x \leq 2 \wedge 1 \leq y < 3)) \overset{\varepsilon}{\longmapsto} (Q, (1 \leq y - x \leq 2))
\end{aligned}
$$

Thus, we have that $(X, (x = y = 0)) \longmapsto^\star (Q, (1 \leq y - x < 2))$.

The following theorem shows that the symbolic semantics of processes is a full characterisation of the concrete semantics.

Theorem 1 (Correctness of Symbolic Semantics). *Assume P^0, $P^f \in \mathcal{P}$, D^0, $D^f \in \mathcal{B}(\mathcal{C})$, and u^0, u^f are clock assignments.*

– *(Soundness) whenever* $(P^0, \{u^0\}) \longmapsto^\star (P^f, D^f)$ *then* $(P^0, u^0) \leadsto^\star (P^f, u^f)$ *for all* $u^f \in D^f$
– *(Completeness) whenever* $(P^0, u^0) \leadsto^\star (P^f, u^f)$ *then* $(P^0, \{u^0\}) \longmapsto^\star (P^f, D^f)$ *for some* D^f *and* $u^f \in D^f$

where $\{u^0\}$ *denotes the clock constraint with* u^0 *being the only solution.*

Proof. We prove both soundness and completeness by induction on the length of transition sequences. First, we show that all transition sequences $(P^0, u^0) \leadsto^n (P^n, u^n)$ can be assumed (without loss of generality) to have alternating actions [4] , i.e. the form $(P^0, u^0) \overset{d_1}{\leadsto} (P^0, u^1) \overset{a_2}{\leadsto} (P^2, u^2) \overset{d_3}{\leadsto} \ldots \overset{a_{n-1}}{\leadsto} (P^{n-1}, u^{n-1}) \overset{d_n}{\leadsto} (P^{n-1}, u^n)$, $d_i \in \mathbb{R}_+$ and $a_i \in \mathcal{A}ct$. Neighboring action-transition in the form $\ldots \overset{a_i}{\leadsto} (P^i, u^i) \overset{a_{i+1}}{\leadsto} \ldots$ can be transformed to $\ldots \overset{a_i}{\leadsto} (P^i, u^i) \overset{0}{\leadsto} (P^i, u^i) \overset{a_{i+1}}{\leadsto} \ldots$ by inserting 0-delay transitions, and neighboring delay-transitions in the form $\ldots \overset{d_i}{\leadsto} (P^i, u^i) \overset{d_{i+1}}{\leadsto} (P^i, u^{i+1}) \ldots$ can be transformed to $\ldots \overset{d_i + d_{i+1}}{\leadsto} (P^i, u^{i+1}) \ldots$.

(Soundness) Assume $(P^0, \{u^0\}) \longmapsto^n (P^n, D^n) \overset{\sigma}{\longmapsto} (P^{n+1}, D^{n+1})$. By induction $(P^0, u^0) \leadsto^n (P^n, u^n)$ and for all $u^n \in D^n$. We need to prove for all $u^{n+1} \in D^{n+1}$, $(P^n, u^n) \overset{\sigma}{\leadsto} (P^{n+1}, u^{n+1})$ for some $u^n \in D^n$. There are two cases since $\sigma \in (\mathcal{A}ct \cup \{\varepsilon\})$:

– $(\sigma \in \mathcal{A}ct)$ From $(P^n, D^n) \overset{a}{\longmapsto} (P^{n+1}, D^{n+1})$ and Definition 6 we have $P^n \overset{g\,a\,r}{\longrightarrow} P^{n+1}$ and $D^{n+1} = r(g \wedge D^n)$. Further, due to Definition 5, we have $D^{n+1} = \{r[u] \mid u \in D^n \text{ and } g(u)\}$. For state (P^n, u^n), by Definition 4 and $P^n \overset{g\,a\,r}{\longrightarrow} P^{n+1}$, we get a transition $(P^n, u^n) \overset{a}{\leadsto} (P^{n+1}, u^{n+1})$ such that $g(u^n)$. Thus, we have that for all $u^{n+1} \in D^{n+1}$, there is a $u^n \in D^n$ such that $g(u^n)$ and $u^{n+1} = r[u^n]$.
– $(\sigma = \varepsilon)$ From $(P^n, D^n) \overset{\varepsilon}{\longmapsto} (P^{n+1}, D^{n+1})$ and Definition 6 we get $P^n = P^{n+1}$, $D^{n+1} = D^{n\uparrow P^n} = ((D^n \wedge I(P^n))^\uparrow \wedge I(P^n))$. Due to Definition 5 $D^{n+1} = \{u^n \oplus d \mid u^n \in D^n \wedge d \in \mathbb{R}_+ \wedge I(P^n)(u^n) \wedge I(P^n)(u^n \oplus d)\}$. For (P^n, u^n), by Definition 4, we get $(P^n, u^n) \overset{d}{\longrightarrow} (P^n, u^n \oplus d)$ if $I(u^n)$ and $I(u^n \oplus d)$. Thus, for all $u^{n+1} \in D^{n+1}$, there is a $u^n \in D^n$ such that $I(u^n)$, $I(u^{n+1})$ and $u^{n+1} = u^n + d$.

(Completeness) Assume $(P^0, u^0) \leadsto^n (P^n, u^n) \overset{\sigma}{\leadsto} (P^{n+1}, u^{n+1})$. By induction step $(P^0, \{u^0\}) \longmapsto^n (P^n, D^n)$ and $u^n \in D^n$. We need to prove $(P^n, D^n) \overset{\sigma}{\longmapsto} (P^{n+1}, D^{n+1})$ for some D^{n+1} and $u^{n+1} \in D^{n+1}$. There are two cases since $\sigma \in (\mathcal{A}ct \cup \mathbb{R}_+)$:

– $(\sigma \in \mathcal{A}ct)$ From $(P^n, u^n) \overset{a}{\leadsto} (P^{n+1}, u^{n+1})$ and Definition 4 it follows that $P^n \overset{g\,a\,r}{\longrightarrow} P^{n+1}$, $u^{n+1} = r[u^n]$ and $g(u^n)$. By Definition 6 and $P^n \overset{g\,a\,r}{\longrightarrow} P^{n+1}$ we get $(P^n, D^n) \overset{a}{\longmapsto} (P^{n+1}, D^{n+1})$ and $D^{n+1} = r(D^n \wedge g)$. Due to Definition 5 $r(D^n \wedge g) = \{r[u] \mid u \in D^n \wedge g(u)\}$. Thus, $r[u^n] \in D^{n+1}$, that is $u^{n+1} \in D^{n+1}$.
– $(\sigma \in \mathbb{R}_+)$ From $(P^n, u^n) \overset{d}{\leadsto} (P^{n+1}, u^{n+1})$, $d \in \mathbb{R}_+$ and Definition 4 we have $P^n = P^{n+1}$, $u^{n+1} = u^n \oplus d$, $I(P^n)(u^n)$, and $I(P^n)(u^n \oplus d)$. From Definition 5 and 6 we get $(P^n, D^n) \overset{\varepsilon}{\longmapsto} (P^n, D^{n+1})$ and $D^{n+1} = D^{n\uparrow P^n} = \{u \oplus e \mid I(P^n)(u) \wedge I(P^n)(u \oplus e) \wedge u \in D^n \wedge e \in \mathbb{R}_+\}$. From $u^n \in D^n$, $I(P^n)(u^n)$, $I(P^n)(u^n \oplus d)$ and $d, e \in \mathbb{R}_+$ we have that $u^n \oplus d \in D^{n+1}$, that is $u^{n+1} \in D^{n+1}$. □

4 Finite Symbolic Semantics of Processes

While the symbolic semantics of timed automata defined in the previous section is coarser than the concrete semantics, it is not a suitable base for a verification algorithm because it is still infinite. The problem is that the symbolic state-space of a timed automaton is not guaranteed to be finite as the number of clock constraints in each symbolic state is unbounded.

[4] This follows from the Time Continuity Lemma (see e.g. [Yi91]).

In this section we shall develop a symbolic semantics which is guaranteed to yield a finite number of so-called *normalised* symbolic states, but still fully characterises the concrete semantics. Normalisation will be used in the next section to develop a verification algorithm for processes. The idea of normalisation is based on the region graph technique [AD90,ACD90] and similar solutions have been proposed in [Rok93,WT94,DT98].

Example 4. Reconsider the timed automaton A_X of Figure 2. It has the following infinite symbolic transition sequence:

$$
\begin{array}{l}
(X, (x = y = 0)) \stackrel{\varepsilon}{\longmapsto} (X, (x = y \wedge y \leq 1)) \stackrel{\tau}{\longmapsto} \\
(P, (x = y \wedge y \leq 1)) \stackrel{\varepsilon}{\longmapsto} \\
(P, (x = y \wedge y \leq 2)) \stackrel{a}{\longmapsto} \\
(P, (x = 0 \wedge 1 \leq y \leq 2)) \stackrel{\varepsilon}{\longmapsto} \\
(P, (x \leq 2 \wedge 1 \leq y \leq 4 \wedge 1 \leq y - x \leq 2)) \stackrel{a}{\longmapsto} \\
(P, (x = 0 \wedge 2 \leq y \leq 4 \wedge 2 \leq y - x \leq 4)) \stackrel{\varepsilon}{\longmapsto} \\
(P, (x \leq 2 \wedge 2 \leq y \leq 6 \wedge 2 \leq y - x \leq 4)) \stackrel{a}{\longmapsto} \\
(P, (x = 0 \wedge 3 \leq y \leq 6 \wedge 3 \leq y - x \leq 6)) \stackrel{\varepsilon}{\longmapsto} \\
(P, (x \leq 2 \wedge 3 \leq y \leq 8 \wedge 3 \leq y - x \leq 6)) \stackrel{a}{\longmapsto} \ldots
\end{array}
$$

Clearly, the transition sequence in Example 4 yields an infinite number of symbolic states as the number of clock constraints is unbounded in the control-node P. The source of the problem is the upper bound on the y-clock that is gradually increased in the symbolic states of the sequence. The key question is how to prevent clock bounds from growing infinitely, without changing the semantics of the processes.

Normalisation is based on the observation that there is a *maximal constant* $k = 2$ appearing in the guards and location invariants of the automaton A_X. When a clock value have grown beyond 2 in A_X, the exact value does not matter anymore since it can not be distinguished by reachability analysis if the clock values over 2 are not of interest. It follows that a given symbolic state (P, D) of A_X can be extended to include *all* clock values that can not be distinguished from the ones already in D, in the above sense.

4.1 Normalisation of Clock Constraints

In this section we define a *normalisation* operator for clock constraints, which is parametrised with a given constant m. We first need to study the syntactical representation of clock constraints more carefully.

A well-known way to represent the class of constraints $\mathcal{B}(\mathcal{C})$ is to use difference bounded matrices (DBM, see [Bel57,Dil89,BL96]). A DBM is a matrix representation providing a canonical representation of clock constraints. For the purposes of this paper, it suffices to consider a DBM D as a constraint in the form $D = \bigwedge_{j \neq i}^{n} x_i - x_j \leq d_{ij}$, where $x_0 = 0$, and d_{ij} are integer numbers[5]. In general, there are several DBM's describing the same set of solutions. However, it has been shown that for each $D \in \mathcal{B}(\mathcal{C})$ there is a unique DBM, denoted D^C, with the same solution set as D, which is *closed under entailment* in the sense that each d_{ij} can not be strengthened without reducing its solution set [Bel57,Dil89].

We now define a normalisation operator on the DBM-representation of clock constraints.

Definition 7. *Assume $D \in \mathcal{B}(\mathcal{C})$ and a natural number m. The m-normalisation of D, denoted $\mathsf{norm}_m(D)$ is the clock constraint obtained by substituting in D^C: all $d_{ij} > m$ with ∞, all $d_{ij} < -m$ with $-m$, and let all d_{ij} with $|d_{ij}| \leq m$ remain d_{ij}.*

Consider again the solution set A for the case of two clocks, shown in Figure 1. The operation $\mathsf{norm}_m(D)$ is illustrated in Figure 1(d). Intuitively, $\mathsf{norm}_m(D)$ is the clock constraint D where all upper bounds greater than m are removed and all lower bounds greater than m are replaced with m.

[5] For reasons of simplicity and clarity in presentation we will only consider the non-strict orderings in the remainder of the paper. However, the techniques given extends easily to strict orderings.

4.2 Normalised Symbolic Transition Rules

In Section 3.2 we have defined a symbolic semantics of processes, which is based on partitioning the concrete state-space in terms of the clock constraints $\mathcal{B}(\mathcal{C})$. In this section we shall give a *finite* symbolic semantics of processes which is based the normalisation operator on clock constraints. Let $\mathcal{B}_m(\mathcal{C})$ denote the subset of $\mathcal{B}(\mathcal{C})$ with no constants greater than m. We shall see that for a given constant m the normalised state-space is partitioned in terms of the constraints $\mathcal{B}_m(\mathcal{C})$ rather than $\mathcal{B}(\mathcal{C})$. As there are finitely many constraints in $\mathcal{B}_m(\mathcal{C})$ and finitely many control-nodes for a given process, the normalised symbolic state-space is guaranteed to be finite.

Let $\mathrm{M}(P)$ denote the maximal integer constant that appears in the guards and the location invariants of P. For a given timed automaton P and a constant $m > \mathrm{M}(P)$, we define the *normalised symbolic semantics* as a transition system where the set of states are the symbolic states (P, D) with $D \in \mathcal{B}_m(\mathcal{C})$. The normalised transition relation $\Longrightarrow_m$ is defined as follows:

Definition 8 (Normalised Symbolic Transition Rules). *Assume the symbol ε representing delays, $\sigma \in \mathcal{A}ct \cup \{\varepsilon\}$ and m is a natural number. We define*

$$(P, D) \stackrel{\sigma}{\Longrightarrow}_m (P', \mathsf{norm}_m(D')) \quad \textit{if} \quad (P, D) \stackrel{\sigma}{\longmapsto} (P', D')$$

We shall write $(P^0, D^0) \stackrel{n}{\Longrightarrow}_m (P^n, D^n)$ to denote an alternating sequence in the form: $(P^0, D^0) \stackrel{\varepsilon}{\Longrightarrow}_m (P^1, D^1) \stackrel{a_2}{\Longrightarrow}_m (P^2, D^2) \stackrel{\varepsilon}{\Longrightarrow}_m \cdots \stackrel{a_{n-1}}{\Longrightarrow}_m (P^{n-1}, D^{n-1}) \stackrel{\varepsilon}{\Longrightarrow}_m (P^n, D^n)$ for $a_i \in \mathcal{A}ct$, and $(P^0, D^0) \stackrel{\star}{\Longrightarrow}_m (P^f, D^f)$ if $(P^0, D^0) \stackrel{n}{\Longrightarrow}_m (P^n, D^n)$ for some n. □

Thus, the only difference between the normalised symbolic semantics and the symbolic semantics, defined in Definition 6, is that the clock constraints are normalised in the normalised semantics. As all m-normalised constraints belong to $\mathcal{B}_m(\mathcal{C})$, and $\mathcal{P}$ is finite, the normalised symbolic semantics is guaranteed to yield a finite number of symbolic states.

Example 5. Reconsider the infinite symbolic transition sequence shown in Example 4. In the finite symbolic semantics, with $m = 3$, we get the following transition sequence:

$$\begin{array}{l}
(X, (x = y = 0)) \stackrel{\varepsilon}{\Longrightarrow}_m (X, (x = y \wedge y < 1)) \stackrel{\tau}{\Longrightarrow}_m \\
(P, (x = y \wedge y < 1)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P, (x = y \wedge y < 2)) \stackrel{a}{\Longrightarrow}_m \\
(P, (x = 0 \wedge 1 \leq y < 2)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P, (x < 2 \wedge 1 \leq y \wedge 1 \leq y - x < 2)) \stackrel{a}{\Longrightarrow}_m \\
(P, (x = 0 \wedge 2 \leq y \wedge 2 \leq y - x)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P, (x \leq 2 \wedge 2 \leq y \wedge 2 \leq y - x)) \stackrel{a}{\Longrightarrow}_m \\
(P, (x = 0 \wedge 3 \leq y \wedge 3 \leq y - x)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P, (x \leq 2 \wedge 3 \leq y \wedge 3 \leq y - x)) \stackrel{a}{\Longrightarrow}_m \ldots
\end{array}$$

which is the sequence of Example 4 but with all clock constraints normalised.

To establish the correctness of the normalised symbolic semantics we shall need the following properties of the normalisation operator.

Lemma 1. *Assume $D \in \mathcal{B}(\mathcal{C})$ and a natural number m. We have that for all time assignments u with $\max_i(\lceil u(x_i) \rceil) = k$ and $k < m$*

(1) *$u \in D$ if and only if $u \in \mathsf{norm}_m(D)$*
(2) *$u \in (g \wedge D)$ if and only if $u \in (g \wedge \mathsf{norm}_m(D))$*
(3) *$u \in D^{\uparrow}$ if and only if $u \in (\mathsf{norm}_m(D))^{\uparrow}$*
(4) *$u \in \{x\}D$ if and only if $u \in \{x\}(\mathsf{norm}_m(D))$*

where $g \in \mathcal{B}_k(\mathcal{C})$ and $x \in \mathcal{C}$.

Proof. (1) Follows from the fact that the normalisation operator defined in Definition 7 does not affect any time assignments $u \in D^C$ with $\max_i(\lceil u(x_i) \rceil) < m$.

(2) Follows from (1) and the definition of the constraint operation "$\wedge$" in Definition 5.
(3) ($\Rightarrow$) This direction follows from the monotonicity of the $\uparrow$-operator and (1).
($\Leftarrow$) Assume $u \in (\mathsf{norm}_m(D))^{\uparrow}$. By Definition 5 there must exist $u = u' \oplus d$ for $u' \in \mathsf{norm}_m(D)$ and $d \in \mathbb{R}_+$. It follows that $\max_i(\lceil u(x_i) \rceil) < m$ since $\max_i(\lceil u'(x_i) \rceil) < m$, and due to (1) that $u' \in D$ which gives that $u' \oplus d \in D^{\uparrow}$.
(4) ($\Rightarrow$) This follows from the monotonicity of the reset-operator and (1).
($\Leftarrow$) Assume $u \in \{x\}(\mathsf{norm}_m(D))$. By Definition 5 we have that $u(x) = 0$ and that there exists $u' \in (\mathsf{norm}_m(D))$ with $u(y) = u'(y)$ for all $y \neq x$. There are now two cases, either $u'(x) < m$ or $u'(x) \geq m$. First assume $u'(x) < m$. Then $u' \in D$ due to (1) and it follows that $u \in \{x\}D$. For the case $u'(x) \geq m$, assume that there is no $u'' \in D$ with $u''(x) \geq m$ and $u'(y) = u''(y)$ for all $y \neq x$. Then $x \leq l$ for some $l < m$ must be a constraint in D^C. It follows from Definition 7 that $x \leq l$ must also be a constraint in $\mathsf{norm}_m(D)^C$ which contradicts the assumption that $u'(x) \geq m$. □

The following theorem shows how the normalised symbolic semantics characterises the concrete operational semantics defined in the Section 2.2.

Theorem 2 (Correctness of Normalised Symbolic Semantics).
Assume $P^0 \in \mathcal{P}$, $k = \mathrm{M}(P^0)$ and u^0 is a time assignment.

- *(Soundness) whenever $(P^0, \{u^0\}) \Longrightarrow^{\star}_m (P^f, D^f)$ then $(P^0, u^0) \rightsquigarrow^{\star} (P^f, u^f)$ for all $u^f \in D^f$ and $m > \max(k, \max_i(\lceil u^f(x_i) \rceil))$*
- *(Completeness) whenever $(P^0, u^0) \rightsquigarrow^{\star} (P^f, u^f)$ and $m > \max(k, \max_i(\lceil u^f(x_i) \rceil))$ then $(P^0, \{u^0\}) \Longrightarrow^{\star}_m (P^f, D^f)$ for some D^f and $u^f \in D^f$*

where $\{u^0\}$ denotes the clock constraint with u_0 being the only solution.

Proof. We prove both soundness and completeness by induction on the length of the transition sequences.

(Soundness) Assume $(P^0, \{u^0\}) \Longrightarrow^{n}_m (P^n, D^n) \overset{\sigma}{\Longrightarrow}_m (P^{n+1}, D^{n+1})$. By induction we have $(P^0, u^0) \rightsquigarrow^{n} (P^n, u^n)$ for all $u^n \in D^n$ with $\max_i(\lceil u^n(x_i) \rceil) < m$, and we need to prove, for all $u^{n+1} \in D^{n+1}$ with $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$, $(P^n, u^n) \overset{\sigma}{\rightsquigarrow} (P^{n+1}, u^{n+1})$, for some $u^n \in D^n$ such that $\max_i(\lceil u^n(x_i) \rceil) < m$. We have two cases since $\sigma \in \mathcal{A}ct$ or $\sigma = \varepsilon$:

- ($\sigma \in \mathcal{A}ct$) From $(P^n, D^n) \overset{\sigma}{\Longrightarrow}_m (P^{n+1}, D^{n+1})$, Definition 8 and 6 we have $P^n \xrightarrow{g\,a\,r} P^{n+1}$ and $D^{n+1} = \mathsf{norm}_m(r(g \wedge D^n))$. Due to Definition 5, we get $D^{n+1} = \mathsf{norm}_m(\{r[u^n] \mid u^n \in D^n \text{ and } g(u^n)\})$. Further, by Lemma 1(1) and $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$ we have $u^{n+1} \in \{r[u^n] \mid u^n \in D^n \text{ and } g(u^n)\}$. From Definition 4, Lemma 1(2), 1(4) and $P^n \xrightarrow{g\,a\,r} P^{n+1}$ we get a transition $(P^n, u^n) \overset{a}{\rightsquigarrow} (P^{n+1}, r[u^n])$ such that $g(u^n)$ and $\max_i(\lceil u^n(x_i) \rceil) < m$. Thus, we have that for all $u^{n+1} \in D^{n+1}$ with $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$ there is a $u^n \in D^n$ with $\max_i(\lceil u^n(x_i) \rceil) < m$ such that $g(u^n)$ and $u^{n+1} = r[u^n]$.
- ($\sigma = \varepsilon$) From $(P^n, D^n) \overset{\varepsilon}{\Longrightarrow}_m (P^{n+1}, D^{n+1})$, Definition 8 and 6 we get $P^n = P^{n+1}$ and $D^{n+1} = \mathsf{norm}_m(D^{n\uparrow P^n})$. By Lemma 1 and $\max_i(\lceil u^{n+1}(x_i) \rceil) \leq k$ we have $u^{n+1} \in D^{n\uparrow P^n} = (D^n \wedge I(P^n))^{\uparrow} \wedge I(P^n)$, which by Definition 5 has the solution set $(\{u^n \oplus d \mid u^n \in D^n \wedge d \in \mathbb{R}_+ \wedge I(P^n)(u^n) \wedge I(P^n)(u^n \oplus d)\})$. By Definition 4, Lemma 1(3) and (P^n, u^n) we get $(P^n, u^n) \xrightarrow{d} (P^n, u^n \oplus d)$ such that $I(u^n)$, $I(u^n \oplus d)$ and $\max_i(\lceil u^n(x_i) \rceil) < m$. Thus, for all $u^{n+1} \in D^{n+1}$ with $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$, there is a $u^n \in D^n$ with $\max_i(\lceil u^n(x_i) \rceil) < m$ such that $I(u^n)$, $I(u^{n+1})$ and $u^{n+1} = u^n \oplus d$ for some d.

(Completeness) Assume $(P^0, u^0) \rightsquigarrow^{n} (P^n, u^n) \overset{\sigma}{\rightsquigarrow} (P^{n+1}, u^{n+1})$ and $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$. By induction step $(P^0, \{u^0\}) \Longrightarrow^{n}_m (P^n, D^n)$ and $u^n \in D^n$. We need to prove $(P^n, D^n) \overset{\sigma}{\Longrightarrow}_m (P^{n+1}, D^{n+1})$, for some D^{n+1} and $u^{n+1} \in D^{n+1}$. However, from Theorem 1 we have that there is a symbolic transition $(P^n, D^n) \overset{\sigma}{\longmapsto} (P^{n+1}, D_s^{n+1})$ such that $u^{n+1} \in D_s^{n+1}$. Further, from Definition 8 we have $(P^n, D^n) \overset{\sigma}{\Longrightarrow}_m (P^{n+1}, D^{n+1})$ and $D^{n+1} = \mathsf{norm}_m(D_s^{n+1})$. By Definition 7 we have $D^{n+1} \supseteq D_s^{n+1}$ and thus $u^{n+1} \in D^{n+1}$. □

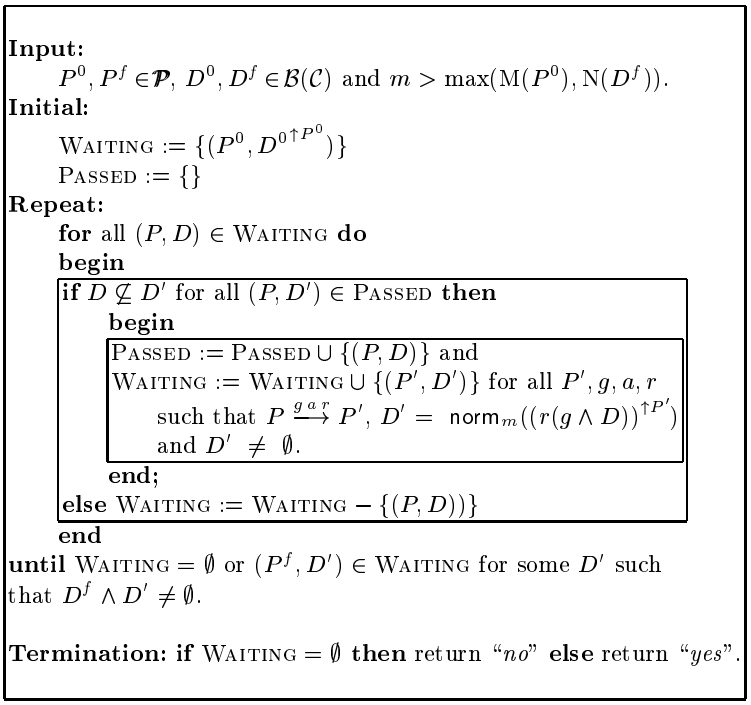

Fig. 3. An Algorithm for Reachability Analysis.

5 Checking Safety Properties of Processes

The language developed in Section 2 can be used to construct abstract models of existing systems or systems to be designed. In this section we discuss how to verify properties of such systems in terms of their abstract models.

5.1 Reachability Analysis

It has been pointed out that the practical goal of verification of real-time systems, in particular safety-critical systems, is often to verify safety properties [Hal93]. These properties are usually formalised as invariant properties in the form $\mathsf{INV}(\neg\varphi)$ read as "φ is invariantly false", where φ describes a certain undesired situation or logical property. For finite-state systems, safety properties can be verified simply by checking all reachable states whether they satisfy φ or not, that is by "reachability analysis".

Let $\mathrm{N}(D)$ denote the the maximal integer constants that appears in the constraints of D. We shall consider the following reachability problem:

Definition 9 (Reachability of Normalised Symbolic States). *Assume* $P^0, P^f \in \mathcal{P}$, $D^0, D^f \in \mathcal{B}(\mathcal{C})$ *and* $k = \max(\mathrm{M}(P^0), \mathrm{N}(D^f))$. *We say that a symbolic state* (P^f, D^f) *is* reachable *from* (P^0, D^0) *if for all* $m < k$, $(P^0, D^0) \Longrightarrow_m^\star (P^f, D)$ *for some* $D \in \mathcal{B}_m(\mathcal{C})$ *and* $D \wedge D^f \neq \emptyset$. □

5.2 An Algorithm for Reachability Analysis

In this section we present an algorithm for forwards reachability analysis[6] of timed automata based on the finite symbolic semantics. To improve the presentation, we shall simply call (P, D) a state instead of a (normalised) symbolic state whenever it is not confusing.

[6] It can easily be adopted to backwards reachability analysis (see [YPD94]).

The algorithm is based on the following idea: Assume that we want to decide whether (P, D) may reach (P', D') in one step (i.e. without passing other control-nodes) or not. The first thing to check is whether it is possible for P to switch to P' directly. If this is not the case, i.e. $P \xrightarrow{g\ a\ r} P'$ for no P', g, a, r, we can immediately conclude that (P', D') is not reachable from (P, D) in one step. Now, assume $P \xrightarrow{g\ a\ r} P'$. To reach (P', D'), there should be clock constraints D^1, D^2 and D^3 such that $D' \wedge D^3 \neq \emptyset$ and $(P, D) \stackrel{\varepsilon}{\Longrightarrow}_m (P, D^1) \stackrel{a}{\Longrightarrow}_m (P', D^2) \stackrel{\varepsilon}{\Longrightarrow}_m (P', D^3)$. Note that D and D' are given. From the normalised symbolic semantics in Definition 8 we get $D^1 = \mathsf{norm}_m(D^{\uparrow P}), D^2 = \mathsf{norm}_m(r(g \wedge D^1)), D^3 = \mathsf{norm}_m(D^{2\uparrow P'})$. That is, $D^3 = \mathsf{norm}_m(\mathsf{norm}_m(r(g \wedge \mathsf{norm}_m(D)^{\uparrow P}))^{\uparrow P'})$.

The algorithm for forwards reachability analysis is shown in Figure 3. It uses two buffers for saving states: PASSED and WAITING, where PASSED holds the set of states that have been examined, and WAITING the set of states that are to be examined next. When the algorithm is started PASSED $= \{\}$ and WAITING $= \{(P^0, D^{0\uparrow P^0})\}$, where $D^{0\uparrow P^0}$ is the largest set of clock assignments that can be reached by idling in P^0. The algorithm then repeatedly examines the states in WAITING. If a state (P, D) found in WAITING that is smaller[7] than a state (P, D') in PASSED, then (P, D) does not need to be examined further. Otherwise, put all the states that are reachable from (P, D) in one step into WAITING to be examined further, and put (P, D) into PASSED. The algorithm will terminate when WAITING is empty (i.e. nothing is left to be examined, and therefore fails to find the final state) or a state (P^f, D') is found, which includes a part of the final state (P^f, D^f) (i.e. $D^f \wedge D' \neq \emptyset$).

We now show that the algorithm is partially correct: Given proper inputs, it always provides the right answer.

Theorem 3 (Partial Correctness). *For all initial states (P^0, D^0) and final states (P^f, D^f):*

1. *whenever the algorithm terminates with answer "*yes*", there exists $(P^0, u^0) \in (P^0, D^0)$, $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ for some $(P^f, u^f) \in (P^f, D^f)$*
2. *whenever the algorithm terminates with answer "*no*", then for all $(P^0, u^0) \in (P^0, D^0)$, $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ for no $(P^f, u^f) \in (P^f, D^f)$*

Proof. Let $\mathrm{SP}_m((P, D))$ denote $\{ (P', D') \mid (P, D) \Longrightarrow^\star_m (P', D') \}$, i.e. the set of all normalised symbolic states reachable from (P, D). We first show that the two sets PASSED and WAITING used in the algorithm satisfies the following invariant property:

$$\mathrm{SP}_m((P^0, D^0)) \;=\; \textsc{Passed} \;\cup \bigcup_{(P,D) \in \textsc{Waiting}} \mathrm{SP}_m((P, D))$$

(**Initial:**) Trivial, as $\mathrm{SP}_m((P^0, D^0)) = \mathrm{SP}_m((P^0, D^{0\uparrow P^0}))$, PASSED $= \emptyset$, and $(P^0, D^0) \stackrel{\varepsilon}{\Longrightarrow}_m (P^0, D^{0\uparrow P^0})$ by Definition 6.
(**Repeat:**) Assume the invariant property holds for the current values of PASSED and WAITING, $(P, D) \in$ WAITING, and $P \neq P^f$ or $D \wedge D^f = \emptyset$. Assume also that there is no $(P, D') \in$ PASSED such that $D \subseteq D'$. The algorithm then updates PASSED and WAITING to PASSED $\cup \{(P, D)\}$ and WAITING $- \{(P, D)\} \cup \{(P', D') \mid P \xrightarrow{g\ a\ r} P'$ and $D' = \mathsf{norm}_m((r(D \wedge g))^{\uparrow P'})\}$. This does not modify the set of states on the r.h.s. of the invariant property, since the only symbolic state (P, D) removed from WAITING is added to PASSED and all successor states of (P, D) are added to WAITING.

Now assume $D \subseteq D'$ for some $(P, D') \in$ PASSED. Then $\mathrm{SP}_m((P, D)) \subseteq \mathrm{SP}_m((P, D'))$ and thus (P, D) does not have to be further explored.
(**Termination:**) Trivial as PASSED and WAITING are not updated. Note that if the algorithm terminates on the criteria WAITING $= \emptyset$, then $\mathrm{SP}_m((P^0, D^0)) =$ PASSED, i.e. PASSED holds all reachable normalised symbolic states.

We now establish that the criteria (1) and (2) for partial correctness holds.

[7] The symbolic state (P, D) is smaller than (P', D') if $P = P'$ and $D \subseteq D'$.

(1) The algorithm terminates with "*yes*" whenever there exists $(P, D) \in$ WAITING, $P = P^f$ and $D \wedge D^f \neq \emptyset$. From the above invariant we have $(P^0, D^0) \Longrightarrow_m^\star (P^f, D)$. It follows from Theorem 2 that $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ for some $u^0 \in D^0$ and $u^f \in D^f$.
(2) When the algorithm terminates with "*no*" PASSED $= \text{SP}_m((P^0, D^0))$ and no $(P, D) \in$ PASSED is found such that $P = P^f$ and $D \wedge D^f \neq \emptyset$. By Theorem 2 we have that $(P_0, u_0) \rightsquigarrow^\star (P_f, u_f)$ for no $u^0 \in D^0$ and $u^f \in D^f$. □

Finally, we prove total correctness of the algorithm: Given proper inputs, it always terminates with an answer.

Theorem 4 (Total Correctness). *For all initial states (P^0, D^0) and final states (P^f, D^f), the algorithm always terminates with an answer which is either "*yes*" or "*no*".*

Proof. The theorem follows from the following two lemmas.

Lemma 2 (Closure Property of Clock Constraints). *Assume that $\mathcal{C}$ is a set of clocks, $x \in \mathcal{C}$, and $D, D' \in \mathcal{B}(\mathcal{C})$ with solution sets A and A'. Then there exist clock constraints D^1, D^2, D^3 $\in \mathcal{B}(\mathcal{C})$ with solution sets $A \wedge A'$, $A^\uparrow$, and $\{x\}A$*

Proof. See [Dil89]. □

Lemma 3. *Assume $D \in \mathcal{B}(\mathcal{C})$ and a non-negative integer number m. Then $\mathsf{norm}_m(D) \in \mathcal{B}_m(\mathcal{C})$, where $\mathcal{B}_m(\mathcal{C})$ is the subset of $\mathcal{B}(\mathcal{C})$ with no constants greater than m.*

Proof. Follows from Definition 7. □

Thus, due to Lemma 2 the class of clock constraints is closed under the operations on constraints, used in the algorithm. Further, from Lemma 3 the number of constraints manipulated by the algorithm is bounded. Since the number of control-nodes of P_0 is finite, we have that the algorithm is guaranteed to terminate with an answer. □

6 Examples

We have implemented the algorithm presented in the previous section in the two verification tools TAB and UPPAAL. The tool TAB, was developed in 1993 based on the symbolic and on-the-fly backwards reachability analysis algorithm for timed automata presented in the conference version of this paper [YPD94]. TAB is implemented in a constraint solver developed at the Swedish Institute of Computer Science (SICS) called Prolog Constraint Solver [Nil93].

In this section, we present two examples which have been analysed in the successor of TAB, called UPPAAL [LPY95,LPY97]. The first version of UPPAAL, implemented in C^{++} and with efficient operations on constraints, was finished in 1995 [LPY95]. In addition to clock variables the UPPAAL-model, which is based on the model of networks of timed automata presented in this paper, has integer data variables. These variables do not change their values at the delay-transitions as the clock variables do; they can only be assigned to values from finite domains, and therefore they will not cause infinite-stateness.

6.1 Fischer's Mutual Exclusion Protocol

The protocol was proposed originally by Fischer and described by Lamport [AL92]. It is to guarantee mutual exclusion in a concurrent system consisting of several processes using a variable shared among the processes. Each of the processes is assumed to have a local clock. The idea behind the protocol is that the timing constraints on the local clocks ensure that only one process can set the shared variable to its own process number, then later if the shared variable is still equal to its own number, enter the critical section.

$$
\begin{aligned}
P_i &\stackrel{def}{=} A_i \\
A_i &\stackrel{def}{=} ((v = 0), \tau, \{x_i\}).B_i \\
B_i &\stackrel{def}{=} (x_i < \mathbf{c}) \gg (\mathtt{tt}, \tau, \{v := i, x_i\}).C_i \\
C_i &\stackrel{def}{=} ((v = i, x_i > \mathbf{c}), \tau, \{\}).CS_i \\
CS_i &\stackrel{def}{=} \mathsf{nil}
\end{aligned}
$$

Fig. 4. Formal Description of Fischer's Protocol.

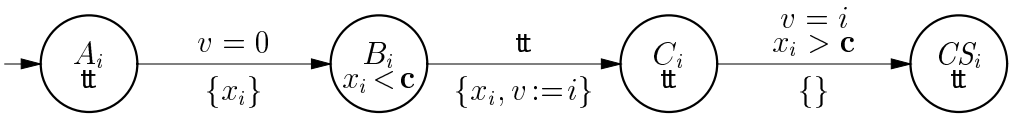

Fig. 5. Fischer's Mutual Exclusion Protocol.

Assume a concurrent system with n processes $P_1, \ldots, P_n$. We use x_i to model the local clock for each process P_i. The formal description of P_i is given in Figure 4, and the timed automaton described by P_i is illustrated in Figure 5 [8].

This is a simplified version of the original protocol and has been studied by researchers, e.g. [AL92,Sha93], which permits only one process to enter the critical section and never exits it. Recovery actions from failure to enter the critical section are omitted. However, the protocol can be extended to an actual mutual exclusion algorithm.

The processes P_i may be in either of the four local states A_i, B_i, C_i, CS_i. Initially, all processes are in their A-states and the shared variable v is initially 0. A process, P_i, that tries to enter the critical section changes state from A_i to B_i if it sees $v = 0$. In B_i, it will move to C_i before the clock x_i proceeds to $\mathbf{c}$, and in doing so, reset the clock x_i (i.e. $x_i := 0$) and assign v to its own process number (i.e. $v := i$). From C_i, it can move to the critical section CS_i if v is still equal to its process number (i.e. $v = i$) when the clock value of x_i is larger than $\mathbf{c}$.

Intuitively, the protocol behaves as follows: The constraints on the shared variable v ensure that a process must reach B-node before any process reach C-node; otherwise, it will never move from A-node to B-node. The timing constraints on the clocks ensure that all processes in C-node must wait until all processes in B-node reach C-node. The last process that reached C-node and set v to its own process number gets the right to enter its critical section. In fact, the protocol will guarantee mutual exclusion for any non-zero constant $\mathbf{c}$.

We need to check that the mutual exclusion property is satisfied, i.e. there will never be more than one process which may reach the critical section, CS_i. The requirement can be formalised as follows: The concurrent system, with an initial state where the control-node is $A_1 \mid \ldots \mid A_n$ and arbitrary variable assignment, will never reach a state where the control-node is in the form

$$S_1 \mid \ldots \mid CS_k \mid \ldots \mid CS_l \mid \ldots \mid S_n$$

for some $k, l \leq n$ and $S_i \in \{A_i, B_i, C_i, CS_i\}$.

We have used UPPAAL and verified the system consisting of 12 processes and with $\mathbf{c} = 1$, which satisfies the property [9].

[8] In Figure 5 and 7 we adopt the convention that when a transition is not labelled with an action, it means that the transition is an internal one, that is τ.

[9] UPPAAL version 2.18.3 consumed 8376 seconds of CPU time and 265 MB of memory on a Pentium Pro 200 MHz machine running Redhat Linux 5.0.

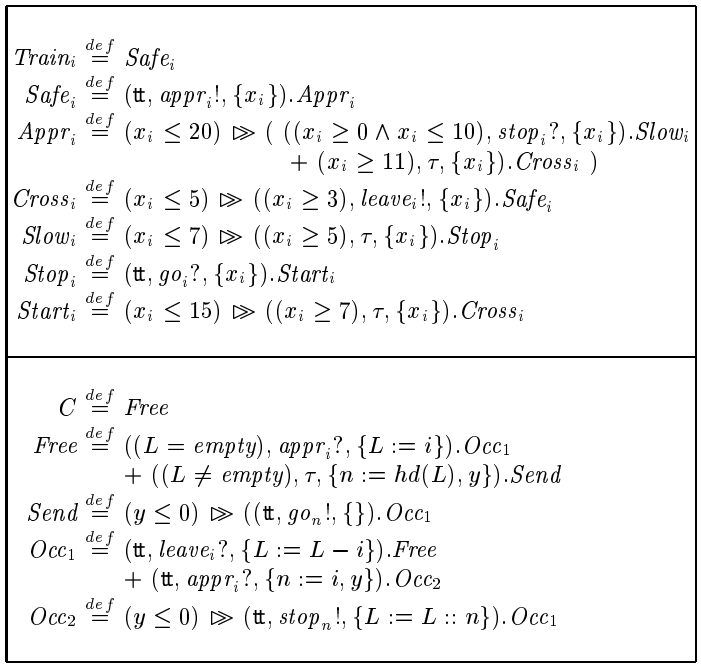

Fig. 6. Formal Description of the Railway Control System.

6.2 A Railway Control System

We consider a railway control system to automatically control trains passing a critical point such as a bridge. The idea is to use a computer to guide trains from several tracks crossing a single bridge instead of building many bridges. An obvious safety-property of such a system is to avoid the situation where more than one train are crossing the bridge at the same time.

Assume that the whole system consists of n trains and a controller. We model the system by the following process:

$$(C \mid Train_1 \mid \ldots \mid Train_n) \setminus A$$

where $Train_i$ describe the behavior of the trains, C describes the behavior of the controller, and $A = \{appr_i, stop_i, leave_i, go_i\}$ is the set of internal channel names (or signals) between the trains and the controller.

To describe timing constraints, we use clocks y and x_i to model the local time of the controller and the trains respectively. The controller uses a list L for the trains waiting to cross the bridge. The formal descriptions of $Train_i$'s and C are given in Figure 6 and illustrated in Figure 7.

Intuitively, when a train, $Train_i$, approaches the bridge it sends a signal to the controller within a certain distance. If the bridge is occupied the controller immediately sends a stop signal $stop_i$ to prevent the train from entering the bridge. Otherwise, if the approaching train does not receive a stop signal within 10 time units, it will start to cross the bridge within 20 time units (but it will take at least 10 time units for a train to enter the bridge). The crossing train is assumed to leave the bridge within 3 to 5 time units; a stopped train will slow down and eventually stop after some delay. When the bridge is free again and the controller signals (by sending go_i) the first train in the waiting list to cross.

Assume that the system is started with the following control-node:

$$(Free \mid Safe_1 \mid \ldots \mid Safe_n)$$

and all clocks are initialised to 0.

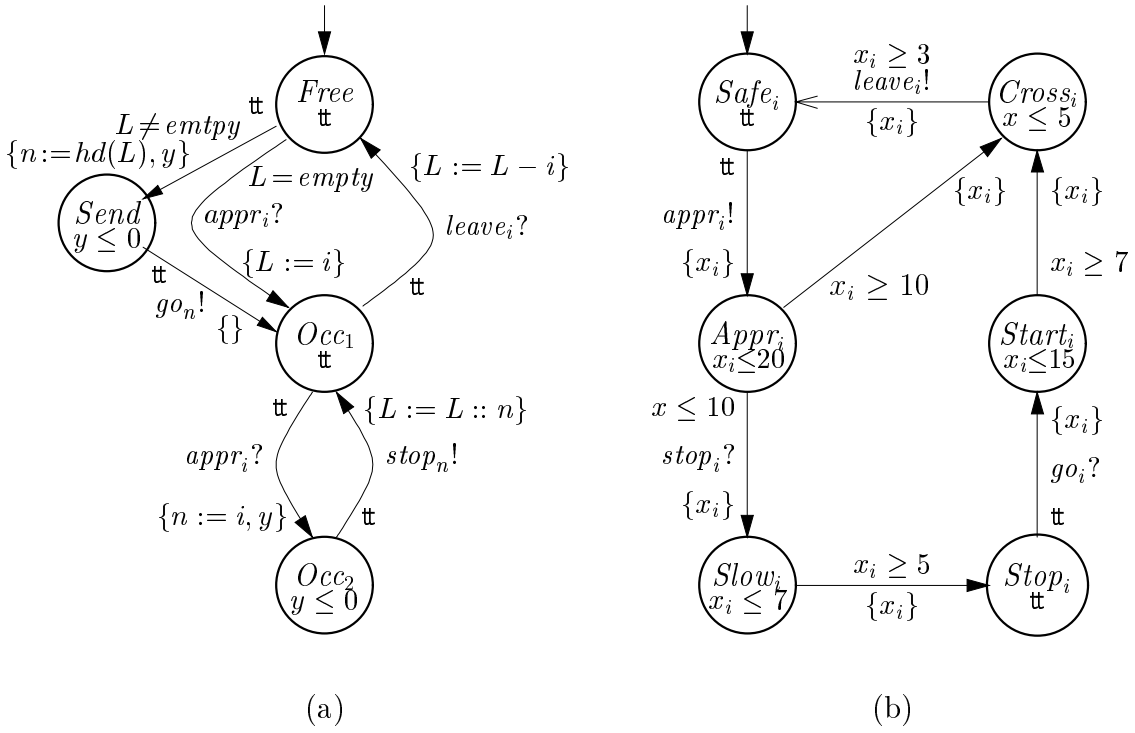

Fig. 7. (a) Controller, (b) Train.

We need to guarantee that the system will never reach a state where two trains are in node *Cross* (the clocks may have any values). That is, a state in the form:

$$(S_i \mid T_1 \mid \ldots \mid Cross_k \mid \ldots \mid Cross_l \mid \ldots \mid T_n)$$

for some $k, l \leq n$, $S_i \in \{Free, Send, Occ_1, Occ_2\}$ and $T_i \in \{Safe_i, Appr_i, Slow_i, Stop_i, Start_i\}$. We have verified that a system consisting of 6 trains satisfies the safety-requirement [10].

7 Conclusion

We have presented an algebra of processes with clocks, which extends timed automata with algebraic operators. The algebra may serve as a formal description language for real-time communicating systems modelled as networks of timed automata. In particular, a parallel composition operator is introduced for timed automata to model communication and concurrency. The operators can be used to construct complex system descriptions in terms of simpler ones.

We have also presented a symbolic on-the-fly reachability analysis algorithm for the description language, based on constraint-solving techniques. The algorithm is proved to be sound (i.e. always provides the right answer) and complete (i.e. always terminates with an answer). It has been implemented in two automatic verification tools for checking safety properties of real-time systems: TAB and UPPAAL. In this paper, as examples, we apply UPPAAL to verify safety properties of a version of Fischer's mutual exclusion protocol and a railway control system.

There have been many proposals for verifying timed systems. However, most of them are intended to construct the whole state-space of a system or to obtain more efficient model-checking algorithms with respect to a real-time temporal logic, or to check equivalences between abstract

[10] UPPAAL version 2.18.3, installed on the same machine as in the previous example, consumed 143 MB of memory and 3019 seconds of CPU time.

specifications. We believe that the practical goal of verifying real-time systems, in particular safety-critical systems is to check simple logical properties, which can be done without constructing the whole state-space. We are of the opinion that our approach is simpler as it is based directly on constraint-solving techniques and can be efficient in verifying systems consisting of many components as it explores (and generates) only the reachable part of the whole state-space.

References

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for Real-Time Systems. In *Proc. of Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, June 1990.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of Timed Transition System. In *Proc. of CONCUR '92, Theories of Concurrency: Unification an d Extension*, number 630 in Lecture Notes in Computer Science, 1992.

[AD90] Rajeev Alur and David Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 322–335, July 1990.

[AL92] Martin Abadi and Leslie Lamport. An Old-Fashioned Recipe for Real Time. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, number 600 in Lecture Notes in Computer Science, 1992.

[Bel57] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[BL96] Johan Bengtsson and Fredrik Larsson. UPPAAL a Tool for Automatic Verification of Real-time Systems. Master's thesis, Uppsala University, 1996. Available as `http://www.docs.uu.se/-docs/rtmv/bl-report.pdf`.

[Cer92] Karlis Cerans. Decidability of Bisimulation Equivalences for Parallel Timer Processes. In *Proc. of CAV'92*, number 663 in Lecture Notes in Computer Science, Berlin, 1992. Springer–Verlag.

[CGL93] Karlis Cerans, Jens Chr. Godskesen, and Kim G. Larsen. Time Modal Specification – Theory and Tools. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science. Springer–Verlag, 1993.

[DB96] Pedro R. D'Argenio and Ed Brinksma. A Calculus for Timed Automata. In Bengt Jonsson and Joachim Parrow, editors, *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1135 in Lecture Notes in Computer Science, pages 110–129. Springer–Verlag, 1996.

[Dil89] David Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer–Verlag, 1989.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 313–329. Springer–Verlag, 1998.

[Hal93] Nicolas Halbwachs. Delay Analysis in Synchronous Programs. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[HLY92] U. Holmer, K.G. Larsen, and W. Yi. Decidability of bisimulation equivalence between regular timed processes. In *Proc. of Workshop on Computer Aided Verification*, number 575 in Lecture Notes in Computer Science, Berlin, 1992. Springer–Verlag.

[HNSY92] Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. In *Proc. of IEEE Symp. on Logic in Computer Science*, 1992.

[Hol91] Gerard Holzmann. *The Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.

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

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, Englewood Cliffs, 1989.

[MT91] F. Moller and C. Tofts. Relating Processes with Respect to Speed. Technical Report ECS–LFCS–91–143, Department of Computer Science, University of Edinburgh, 1991.

[Nil93] Martin Nilsson. Piecewise Linear Constraints and Entailment. Technical report, Swedish Institute of Computer Science, August 1993.

[Rok93] Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits.* PhD thesis, Stanford University, 1993.

[Sha93] N. Shankar. Verification of Real-Time Systems Using PVS. In *Proc. of the* 5th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science. Springer–Verlag, 1993.

[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems.* PhD thesis, Stanford University, November 1994.

[Yi91] Wang Yi. *A Calculus of Real Time Systems.* PhD thesis, Department of Computer Science, Chalmers University of Technology, 1991.

[YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the* 7th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.