Chapter 4

Model Checking Timed Automata

4.1. Introduction

Currently, formal verification of reactive, critical or embedded systems is a crucial problem, and automatic verification, more specifically *model checking*, has been widely developed during the last 20 years (see [CLA 99, SCH 01] for surveys). In this approach, we build a formal model $\mathcal{M}$ (e.g. an automaton, Petri net, etc.) describing the behavior of the system under verification; the correctness property Φ is stated with a formal specification language (e.g. temporal logic), and then a model-checker is used to automatically decide whether $\mathcal{M}$ satisfies Φ or not.

Often, it is necessary to consider real-time aspects: quantitative information about time elapsing has to be handled explicitly. This can be the case when describing a particular behavior (for instance, a time-out) or stating a complex property (for example, "the alarm has to be activated *within at most 10 time units* after a problem has occurred"). In 1990, Alur and Dill have proposed *timed automata* as a model to represent the behavior of real-time systems [ALU 90, ALU 94a]. This formalism extends classical automata with a set of real-valued variables – called clocks – that increase synchronously with time, and associates guards (specifying when, i.e. for which values of the clocks, the transition can be performed) and update operations (to be applied when the transition is performed) with every transition. Thanks to these clocks, it becomes possible to express constraints over delays between two transitions.

Temporal logics have also been extended to deal with real-time constraints. For example, the modalities of the classical CTL logic (computation tree logic [CLA 81])

Chapter written by Patricia BOUYER and François LAROUSSINIE.

have been adapted to handle quantitative constraints over time elapsing [ALU 94c, ALU 93a, ACE 02].

Finally, model checking algorithms have been developed [ALU 93a, HEN 94, LAR 95b], and a lot of research has been done on the timed verification algorithmics: efficient data-structures, on-the-fly algorithms, compositional methods, etc. have been proposed. Timed model checkers have also been developed [YOV 97, LAR 97b] and are applied to industrial case studies [TRI 98, BEN 02]. Timed model checking is clearly an active research topic.

In this chapter, we present the classical timed automata model. We explain the main characteristics of this model, and describe the famous region graph technique that is a crucial construction to obtain the decidability of many verification problems in this framework. We also mention several possible extensions of timed automata and several interesting subclasses. Finally, we describe algorithmic aspects and the basic data-structure that is used to implement verification algorithms, and we present the Uppaal tool [LAR 97b].

4.2. Timed automata

Timed automata have been proposed by R. Alur and D. Dill in the 1990s [ALU 90, ALU 94a] as a model for real-time systems. A timed automaton is a classical finite automaton which can manipulate clocks, evolving continuously and synchronously with absolute time. Each transition of such an automaton is labeled by a constraint over clock values (also called guard), which indicates when the transition can be fired, and a set of clocks to be reset when the transition is fired. Each location is constrained by an invariant, which restricts the possible values of the clocks for being in the state, which can then enforce a transition to be taken. The time domain can be $\mathbb{N}$, the set of non-negative integers, or $\mathbf{Q}$, the set of non-negative rationals, or even $\mathbf{R}$, the set of non-negative real numbers. In this chapter, we choose $\mathbf{R}$ as the time domain, but most results are unchanged when considering $\mathbf{Q}$ or $\mathbb{N}$.

4.2.1. *Some notations*

Let X be a finite set of variables, called clocks, taking values in $\mathbf{R}$. A (clock) valuation v over X is a function $v : X \to \mathbf{R}$ which associates with every clock x its value $v(x) \in \mathbf{R}$. We denote by $\mathbf{R}^X$ the set of clock valuations over X. Given a real $d \in \mathbf{R}$, we write $v + d$ for the clock valuation associating with clock x the value $v(x) + d$, If r is a subset of X, $[r \leftarrow 0]v$ is the valuation v' such that $v'(x) = 0$ if $x \in r$, and $v'(x) = v(x)$ otherwise.

We write $\mathcal{C}(X)$ for the set of clock constraints over X, i.e., the set of Boolean combinations of atomic constraints of the form $x \bowtie c$ with $x \in X$, $\bowtie \in \{=, <, \leqslant, >, \geqslant\}$ and $c \in \mathbb{N}$. We note by $\mathcal{C}_{\leqslant}(X)$ the restriction of $\mathcal{C}(X)$ to positive Boolean combinations only containing constraints of the form $x \leqslant c$ or $x < c$. We interpret clock constraints over clock valuations: a valuation v satisfies the atomic constraint $x \bowtie c$ whenever $v(x) \bowtie c$; the extension to general constraints is then immediate and natural. When a valuation v satisfies a constraint g, we write $v \models g$.

4.2.2. *Timed automata, syntax and semantics*

The formal definition of a timed automaton is as follows.

DEFINITION 4.1.– *A* timed automaton $\mathcal{A}$ *is a tuple* $(L, \ell_0, X, \mathsf{Inv}, T, \Sigma)$ *where:*

– *L is a finite set of control states, also called locations,*

– *$\ell_0 \in L$ is the initial location,*

– *X is a finite set of clocks,*

– *$T \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ is a finite set of transitions: $e = \langle \ell, g, a, r, \ell' \rangle \in T$ represents a transition from ℓ to ℓ', g is the guard of e, r is the set of clocks that is reset by e, and a is the action of e. We also write $\ell \xrightarrow{g,a,r} \ell'$ for e,*

– *$\mathsf{Inv} : L \to \mathcal{C}_{\leqslant}(X)$ associates with each location an invariant,*

– *Σ is an alphabet of actions.*

An example of timed automaton is given in Figure 4.2.

A state, or configuration, of a timed automaton is a pair $(\ell, v) \in L \times \mathbf{R}^X$ where ℓ is the current location and v is the clock valuation. The semantics of a timed automaton is given as a timed transition system with action transitions (labeled with elements of Σ) and delay transitions (labeled with real numbers representing the delay). This is stated more precisely as follows.

DEFINITION 4.2.– *A* timed transition system *(TTS) is a tuple $\mathcal{S} = (S, s_0, \to, \Sigma)$ where S is a (possibly infinite) set of states, $s_0 \in S$ is the initial state and $\to \subseteq S \times (\Sigma \cup \mathbf{R}) \times S$ is the transition relation. Moreover, the relation $\to$ satisfies the three following conditions: (1) if $s \xrightarrow{0} s'$, then $s = s'$, (2) if $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$ with $d, d' \in \mathbf{R}$, then $s \xrightarrow{d+d'} s''$, and (3) if $s \xrightarrow{d} s'$ with $d \in \mathbf{R}$, then for all $0 \leqslant d' \leqslant d$, there exists $s'' \in S$ such that $s \xrightarrow{d'} s''$ and $s'' \xrightarrow{d-d'} s'$.*

The three conditions mentioned above are classical in the framework of timed systems, see e.g. [YI 90], they simply express that the time is continuous and deterministic.

Classically, an execution in a TTS is a sequence of consecutive transitions. A state $s \in S$ is said to be reachable in $\mathcal{S}$ if there exists an execution from s_0 to s.

DEFINITION 4.3.– *Let $\mathcal{A} = (L, \ell_0, X, \mathsf{Inv}, T, \Sigma)$ be a timed automaton. The semantics of $\mathcal{A}$ is defined as the TTS $\mathcal{S}_{\mathcal{A}} = (S, s_0, \to, \Sigma)$ where:*

– $S = L \times \mathbf{R}^X$,

– $s_0 = (\ell_0, v_0)$ *with* $v_0(x) = 0$ *for every* $x \in X$,

– *the transition relation* $\rightarrow$ *is composed of:*

- *action transitions:* $(\ell, v) \xrightarrow{a} (\ell', v')$ *if and only if there exists* $\ell \xrightarrow{g,a,r} \ell' \in T$ *such that* $v \models g$, $v' = [r \leftarrow 0]v$ *and* $v' \models \mathsf{Inv}(\ell')$.

- *delay transitions: if* $d \in \mathbf{R}$, $(\ell, v) \xrightarrow{d} (\ell, v+d)$ *if and only if* $v+d \models \mathsf{Inv}(\ell)$[1].

Informally, the system starts from the initial configuration (location ℓ_0 and all clocks set to zero), and then alternatively takes action transitions if the current clock valuations satisfies the guard (this move is instantaneous and some clocks are then set to zero), and delay transitions which increase all clocks by the same amount of time (clocks are synchronous) while respecting the invariant associated with the current location.

A possible execution of the timed automaton of Figure 4.2 is: $(\ell_0, (0,0)) \xrightarrow{2.67} (\ell_0, (2.67, 2.67)) \xrightarrow{a} (\ell_1, (2.67, 0)) \xrightarrow{1} (\ell_1, (3.67, 1)) \xrightarrow{b} (\ell_2, (3.67, 1)) \cdots$ where the pair $(3.67, 1)$ represents the valuation v such that $v(x) = 3.67$ and $v(y) = 1$.

An execution in a timed automaton can also be seen as a timed word, i.e., a sequence of pairs (action, date). We can then write: $(\ell_0, v_0, t_0) \xrightarrow{a_1} (\ell_1, v_1, t_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (\ell_n, v_n, t_n)$ with $t_i \in \mathbf{R}$, $t_0 = 0$ and $t_{i+1} \geqslant t_i$ for every i. Date t_i corresponds to the time point at which action a_i has been performed. The step $(\ell_i, v_i, t_i) \xrightarrow{a_{i+1}} (\ell_{i+1}, v_{i+1}, t_{i+1})$ corresponds to a delay $t_{i+1} - t_i$ followed by the firing of a transition labeled by a_{i+1}, the valuation v_{i+1} is then obtained from $v_i + (t_{i+1} - t_i)$ by resetting to zero some of the clocks (depending on the transition which has been fired). The associated timed word is then $(a_1, t_1)(a_2, t_2) \cdots$ For instance, the timed word associated with the above-mentioned execution is $(a, 2.67)(b, 3.67) \cdots$.

4.2.3. *Parallel composition*

It is possible to define the parallel composition of timed automata (or of TTSs). For instance, we can define an n-ary synchronization relation with renaming. If this feature is essential for modeling systems, it does not add expressivity power from a theoretical point-of-view: indeed, it is always possible to construct a product automaton having the same behavior as the parallel composition (it is even strongly bisimilar; see section 4.4).

1. Which, given the form of the invariants, is $v + d' \models \mathsf{Inv}(q)$ for every $0 \leqslant d' \leqslant d$.

4.3. Decision procedure for checking reachability

In this section, we describe a construction initially proposed in [ALU 90, ALU 94a] to decide the reachability of a control state in a timed automaton. This construction relies on an abstraction of the behaviors of the timed automaton, so that checking whether a location is reachable in the initial timed automaton is equivalent to checking whether a state (or set of states) is reachable in a finite automaton.

For this aim, an equivalence relation of finite index is defined over the set of configurations of the timed automaton: two configurations s and s' are *equivalent* when any action transition "a" (resp. any delay transition d) enabled from s can be simulated from s' by an action transition "a" (resp. a delay transition d') such that the two resulting configurations are equivalent (and vice versa for s'). Note that precise delays are not respected and the equivalence will only correspond to a *time-abstract bisimulation*. For timed automata, such an equivalence relation (with finite index) always exists, and it is defined as follows. Two configurations (ℓ, v) and (ℓ', v') are equivalent if $\ell = \ell'$ and if $v \equiv_M v'$ (where M is the maximal constant appearing in the automaton). The relation $v \equiv_M v'$ holds whenever for each clock $x \in X$,

1) $v(x) > M \Leftrightarrow v'(x) > M$,

2) if $v(x) \leqslant M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, and $\Big(\{v(x)\} = 0 \Leftrightarrow \{v'(x)\} = 0\Big)$[2],

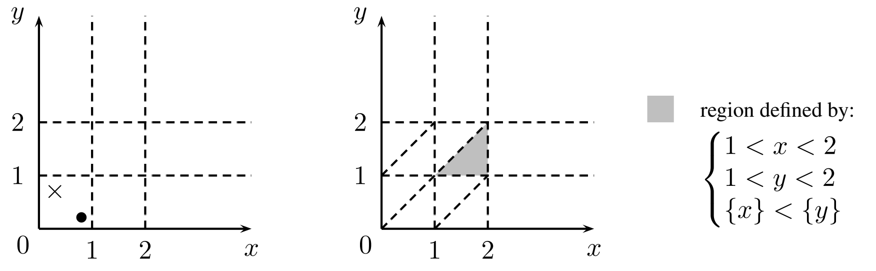
and for each pair of clocks (x, y),

3) if $v(x) \leqslant M$ and $v(y) \leqslant M$, then $\{v(x)\} \leqslant \{v(y)\} \Leftrightarrow \{v'(x)\} \leqslant \{v'(y)\}$.

Intuitively, the two first conditions express that two equivalent valuations satisfy exactly the same clock constraints of the timed automaton. The last condition ensures that from two equivalent configurations, letting time elapse will lead to the same integral values for the clocks, in the very same order. The equivalence $\equiv_M$ is called the *region equivalence*, and an equivalence class is then called a *region*.

We illustrate this construction in Figure 4.1 in the case of two clocks x and y, and the maximal constant is supposed to be 2. The partition depicted in Figure 4.1(a) respects all constraints defined with integral constants smaller than or equal to 2, but the two valuations $\bullet$ and $\times$ are not equivalent due to time elapsing (item 3 above): indeed, if we let some time elapse from the valuation $\bullet$, we will first satisfy the constraint $x = 1$ and then $y = 1$, while it will be the converse for the valuation $\times$. Thus, the possible behaviors from $\bullet$ and $\times$ are different. Condition 3 refines the partition of Figure 4.1(a) by adding diagonal lines (that somehow represent time elapsing), and the resulting partition is given on Figure 4.1(b) and is a time-abstract bisimulation.

2. $\lfloor \alpha \rfloor$ represents the integral part of α whereas $\{\alpha\}$ represents its fractional part.



y
2
1
0
1
2
x
region defined by:
$\begin{cases} 1 < x < 2 \\ 1 < y < 2 \\ \{x\} < \{y\} \end{cases}$

(a) Partition respecting 1) and 2) (b) Partition respecting 1), 2) and 3)

Figure 4.1. *Region partitioning for two clocks and maximal constant* 2

From the initial timed automaton and this equivalence relation, we build a finite automaton as follows: the states of the automaton are pairs (ℓ, R) where ℓ is a location of the timed automaton and R a region; the transitions are $(\ell, R) \xrightarrow{a} (\ell', R')$ if there exists a transition $\ell \xrightarrow{g,a,r} \ell'$ in $\mathcal{A}$, a valuation $v \in R$, and $t \geqslant 0$ such that $v+t \models \mathsf{Inv}(\ell)$, $v+t \models g$, $[r \leftarrow 0](v+t) \models \mathsf{Inv}(\ell')$ and $[r \leftarrow 0](v+t) \in R'$.

The resulting finite automaton $\mathcal{R}_{\mathcal{A}}$ is called the *region automaton* associated with the initial timed automaton. The fundamental property of this finite automaton is that it recognizes exactly the set of words $a_1a_2\cdots$ such that there exists a timed word $(a_1,t_1)(a_2,t_2)\cdots$ recognized by the initial timed automaton. Hence, given a timed automaton $\mathcal{A}$ and its region automaton $\mathcal{R}_{\mathcal{A}}$, we can reduce the emptiness check for the timed language accepted by $\mathcal{A}$ (or equivalently the reachability checking of a location of $\mathcal{A}$) to a reachability problem in $\mathcal{R}_{\mathcal{A}}$. This produces an algorithm to solve these two problems.

THEOREM 4.1.– *[ALU 94a] Checking the reachability of a location in a timed automaton is a PSPACE-complete problem.*

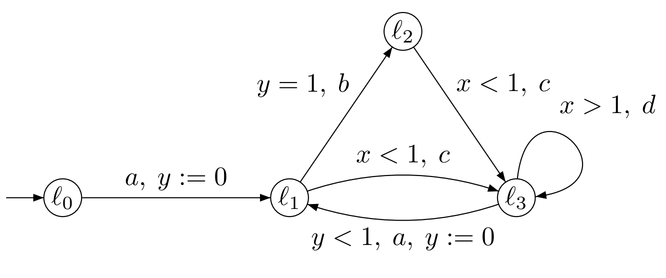


Figure 4.2. *Timed automaton* $\mathcal{A}$

We illustrate the construction of the region automaton on the timed automaton depicted in Figure 4.2 and taken from [ALU 94a]. The corresponding region automaton is depicted in Figure 4.3. In this example, the location ℓ_3 of $\mathcal{A}$ is reachable if and only if one of the states (ℓ_3, R) with a region R is reachable in the finite automaton given in Figure 4.3. In this last automaton, the path $(\ell_0, x{=}y{=}0) \xrightarrow{a} (\ell_1, 0{=}y{<}x{<}1) \xrightarrow{c} (\ell_3, 0{<}y{<}x{<}1)$ leads to location ℓ_3, which implies that, in the timed automaton $\mathcal{A}$, there is an execution $(\ell_0, v_0) \xrightarrow{t_1} (\ell_0, v_0 + t_1) \xrightarrow{a} (\ell_1, v_1) \xrightarrow{t_2} (\ell_1, v_1 + t_2) \xrightarrow{c} (\ell_3, v_2)$ leading to ℓ_3 (for some real numbers t_1 and t_2).

The complexity of the reachability problem, mentioned in the previous theorem, has been originally stated in the papers [ALU 90, ALU 94a]:

– the PSPACE-hardness comes from the fact that we can encode the behavior of a linearly space bounded Turing machine on a given input. Indeed, it is possible to construct a timed automaton in which clock values encode the content of the Turing machine tape along the execution. Notice that such an encoding can be done using only three clocks [COU 92];

– the PSPACE membership is obtained by applying a non-deterministic algorithm which stores the current abstract state of the automaton (location+region) and guesses the next abstract state, until reaching a goal location (or aborting the computation when a counter becomes greater than the size of the region automaton, which is exponential).

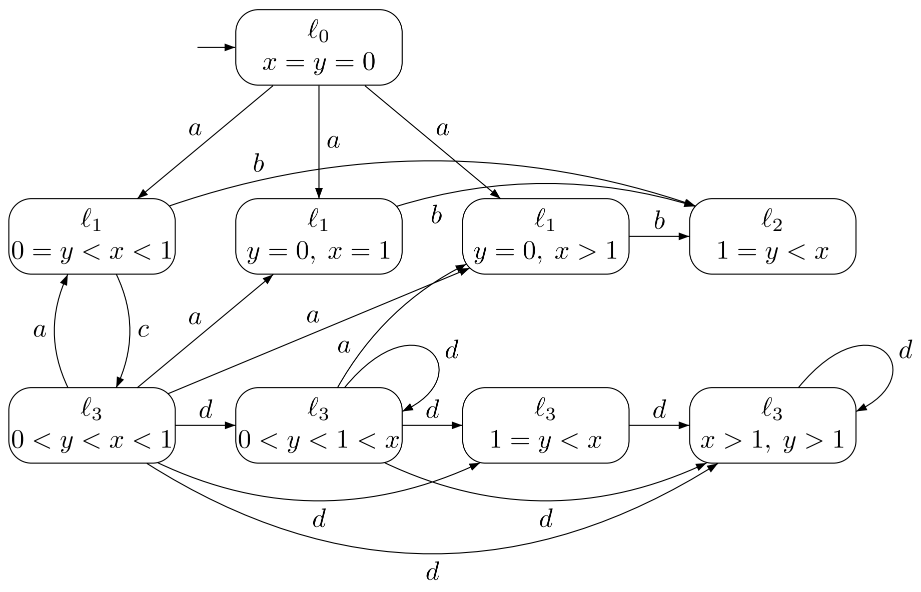


Figure 4.3. *Region automaton associated with* $\mathcal{A}$

4.4. Other verification problems

Reachability is a key problem in verification. Correctness can often be stated as a reachability question: "Is the `BAD` state reachable from the initial configuration?", or "Is it true that any reachable state is either `BLUE` or `RED`?". Nevertheless it is sometimes useful to consider more complex properties on the behavior of the system, and we hence need formal specification languages to state properties. For example, assume we want to express the following timed property:

"The alarm is activated within at most 10 time units after a problem occurs." (4.1)

There are several ways to express such properties over timed systems, and we briefly mention the most classical ones.

4.4.1. *Timed languages*

As mentioned earlier, we can associate a *timed word* with an execution of a timed automaton. It is also possible to consider different acceptance conditions in timed automata (final states, Büchi or Muller conditions, etc.). In this setting, the behavior of a timed automaton $\mathcal{A}$ is seen as a *timed language* $\mathcal{L}(\mathcal{A})$ containing the timed words read over all accepting executions. Now given a property Φ described as a timed language $\mathcal{L}_\Phi$ (for example, the set of words "$(\texttt{request}, t_1), (\texttt{service}, t_2)$" with $t_1 \leqslant t_2 \leqslant t_1 + 10$), the verification problem "does $\mathcal{A}$ satisfy the property Φ?" can be reduced to an inclusion checking over timed languages: is $\mathcal{L}(\mathcal{A})$ included in $\mathcal{L}_\Phi$? In the untimed case, this classical problem is solved by considering the complement of $\mathcal{L}_\Phi$ (i.e. $\mathcal{L}_{\neg\Phi}$) and checking the emptiness of $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}_{\neg\Phi}$. Unfortunately, this approach cannot be used in the timed case because the language $\mathcal{L}_{\neg\Phi}$ is not always expressible with a timed automaton: most timed languages families (of finite words, or infinite words with Büchi or Muller conditions) are not closed under complementation. Indeed the inclusion problem is in general undecidable. Thus, this approach is only possible for restricted classes (e.g. deterministic Muller timed automata).

4.4.2. *Branching-time timed logics*

Temporal logic is a very convenient formalism to specify properties over reactive systems [PNU 77]. They make it possible to express properties over the ordering of events of a system. We can distinguish branching-time temporal logic and linear-time temporal logic: in the former case, formulae are interpreted over states having several possible successors (we can quantify existentially or universally over the different possible futures of a given state). In the latter case, a system is viewed as a set of runs, and formulae express properties over these runs: in such a structure, a state is always considered as *a state along a given run* and then it has a unique successor. These formalisms differ from an expressiveness point of view, and the model checking algorithms are also very different.

The most popular (untimed) branching-time temporal logic is CTL (computation tree logic) [CLA 81]. It contains the modalities "always" (AG), "potentially" (EF), "exists-until" ($\mathsf{E\_U\_}$), and "for-all-until" ($\mathsf{A\_U\_}$). For example, the formula $\mathsf{E}\phi\mathsf{U}\psi$ holds in a state s if and only if there exists a path from s where ψ is true at some position s', and ϕ is true at any position between s and s'. See [EME 91] for a precise introduction to temporal logics for the specification and verification of reactive systems.

There exist several timed extensions of temporal logics. First we can add subscripts with timing constraints to the classical Until operator. Such a constraint is of the form "$\bowtie c$" with $\bowtie \in \{=, <, \leqslant, >, \geqslant\}$ and $c \in \mathbb{N}$. For example, the formula $\mathsf{E}\phi\mathsf{U}_{<c}\psi$ holds in a state s if and only if there exists a run ρ leading to a state s' such that (1) s' satisfies ψ, (2) the duration of ρ is less than c, and (3) any state lying between s and s' along the run ρ satisfies ϕ.

The logic TCTL (for timed CTL) is defined with these extended modalities: it contains the atomic propositions, the Boolean operators and the modalities $\mathsf{E\_U}_{\bowtie c}\_$ and $\mathsf{A\_U}_{\bowtie c}\_$. Thus, Property 4.1 can then be expressed as follows:

$$\mathsf{AG}\Big(\texttt{problem} \Rightarrow \mathsf{AF}_{\leqslant 10}\,\texttt{alarm}\Big)$$

where $\mathsf{AF}_{\leqslant 10}\phi$ is an abbreviation for $\mathsf{A}\ \texttt{true}\ \mathsf{U}_{\leqslant 10}\phi$: along every path, there is a position before 10 time units in which ϕ holds.

There is another way to add timing constraints in CTL. The idea is to consider a new set of clocks – the formula clocks – and to add atomic constraints "$x \bowtie c$" in the logic and a new operator ($\underline{\texttt{in}}$) to reset a given clock to zero [ALU 94c]. This extension is called TCTL_c. The previous property can be expressed as follows with TCTL_c:

$$\mathsf{AG}\Big(\texttt{problem} \Rightarrow \Big(x\ \underline{\texttt{in}}\ \mathsf{AF}\ \big(\texttt{alarm} \wedge (x \leqslant 10)\big)\Big)\Big)$$

where x is a formula clock which is reset to zero when the proposition $\texttt{problem}$ is true, and it is used to ensure that the time elapsed between the problem and the activation of the alarm is less than 10 time units.

This extension with explicit formula clocks allows us to express easily every modality of TCTL. For example, we have the following equivalence when φ and ψ are TCTL formulae:

$$\mathsf{E}\Big(\varphi\mathsf{U}_{\bowtie c}\,\psi\Big) \;\equiv\; x\ \underline{\texttt{in}}\ \mathsf{E}\Big(\varphi\mathsf{U}(\psi \wedge x \bowtie c)\Big)$$

TCTL_c makes it possible to express very precise properties over timed systems. It has been shown recently that it is indeed more powerful than TCTL in the dense time framework [BOU 05b]. For example, the following TCTL_c formula cannot be stated with TCTL:

$$x \underline{\mathtt{in}} \, \mathsf{EF}\Big(P_1 \wedge x < 1 \wedge \mathsf{EG}(x < 1 \Rightarrow \neg P_2)\Big)$$

This formula expresses that it is possible to reach a state s' satisfying P_1 in t time units with $t < 1$ and from then it is possible to avoid P_2 during (at least) $1 - t$ time units.

These specification languages are very convenient to express properties over a timed system. Moreover, verification problems are still decidable.

THEOREM 4.2.– *[ALU 93a] The* TCTL *and* TCTL_c *model checking problems for timed automata are PSPACE-complete.*

The algorithms use the same techniques as for the reachability problem: given a timed automaton $\mathcal{A}$ and a TCTL formula Φ, it is possible to define a region automaton $\mathcal{A}'$ (over the automata clocks and the formula clocks) and a CTL formula Φ' such that $\mathcal{A} \models \Phi$ if and only if $\mathcal{A}' \models \Phi'$. Verifying TCTL formulae over parallel compositions of timed automata can be done with the Kronos tool [YOV 97].

4.4.3. *Linear-time timed logics*

Linear-time temporal logics (as LTL [PNU 81]) can also be extended with timing constraints in the same manner as for the branching-time case. For example, formula $\mathsf{G}(\mathtt{problem} \Rightarrow \mathsf{F}_{\leqslant 10}\mathtt{alarm})$ expresses Property 4.1. The only difference is that such formulae are interpreted over the runs of a timed automaton. By convention we write $\mathcal{A} \models \Phi$ to specify that *every* run of the timed automaton $\mathcal{A}$ satisfies Φ.

In this framework we can mention MTL [KOY 90, ALU 93b] which contains modalities U_I where I is an interval of the form $(l; u)$, $[l; u]$, ... with $l, u \in \mathbb{N} \cup \{\infty\}$. This interval provides the timing constraint in a natural way: formula $P_1 \mathsf{U}_{[3;\infty]} P_2$ is equivalent to $P_1 \mathsf{U}_{\geqslant 3} P_2$, etc. We denote MITL [ALU 96] the fragment of MTL where singular intervals $[c; c]$ are not allowed (and then the modality $\mathsf{U}_{=c}$ is forbidden).

Model checking MTL is undecidable [ALU 96, OUA 06]. Note that it is also possible to consider a different semantic where atomic propositions are interpreted as punctual events occurring at some date: in this case, model checking MTL becomes decidable over finite runs [OUA 05].

The model checking problem for MITL is easier: it is EXPSPACE-complete (with the standard semantics) and even becomes PSPACE-complete as soon as we only consider the modalities $\mathsf{U}_{<c}$ and $\mathsf{U}_{>c}$ (i.e. $\mathsf{U}_{[0;c)}$ and $\mathsf{U}_{(c;\infty)}$) [ALU 96]. Other tractable fragments of MTL have been recently investigated [BOU 07b].

4.4.4. *Timed modal logics*

It is also possible to consider timed extensions of modal logics (see for example the Hennessy-Milner logic [HEN 85]). In this case, we use modalities $\langle a \rangle$ and $[a]$ to deal with the label of transitions. For example, $\langle a \rangle\ \phi$ states that there exists an a-transition leading to a state verifying ϕ, and $[a]\ \phi$ expresses that *every* state reachable via an a-transition satisfies ϕ. These modalities only deal with states reachable in one step. However, it is possible to use fixpoint to express properties over unbounded behaviors [LAR 90, STI 01]. This kind of formalism can also be extended with formula clocks, atomic constraints "$x \bowtie c$" and reset operator $\underline{\texttt{in}}$ as in TCTL_c [LAR 95a]. These logics make it possible to express very precise and subtle properties (e.g. the timed bisimilarity). Model checking these timed modal logics is usually EXPTIME-complete [ACE 02].

4.4.5. *Testing automata*

It is sometimes easy to describe a property to be checked with a timed testing automaton $\mathcal{T}_\phi$. The idea is then to synchronize $\mathcal{T}_\phi$ with the system under verification, and the property checking reduces to some reachability problem of a bad state (when the system does not satisfy the property) or a good state (when the system meets the property) [ACE 98]. The relationship between this approach and the timed modal logics is studied in [ACE 03].

4.4.6. *Behavioral equivalences*

As for the untimed systems, it is also possible to compare timed systems with respect to several behavioral equivalences. For example, we can consider the timed bisimulation: two states s and s' are said to be strongly timed bisimilar when any transition from s can be simulated from s' by a transition with the same label (the same action or the same amount of time) and the successor states have to also be strongly timed bisimilar. This equivalence is very strong: two systems that are strongly bisimilar cannot be distinguished by any temporal or modal logics mentioned previously, they satisfy exactly the same formulae. Deciding whether two timed automata are strongly timed bisimilar is an EXPTIME-complete problem [LAR 00].

Other equivalences can be considered, for example the time-abstract bisimulation (mentioned in section 4.3 concerning the region graph technique) only requires that a delay transition is simulated by another delay transition (however, possibly with another amount of time).

4.5. Some extensions of timed automata

To help model real systems, it might be useful to manipulate a high-level description language. Hence, several extensions of timed automata havebeen considered in other works, and we will present some of them in this section. For each of these extensions, we will be interested in: 1) its decidability, 2) its expressiveness with respect to the original model, 3) its conciseness with respect to the original model. The first item is crucial if we aim at using this extension for modeling real systems. It is also important to have models which can express many systems (item 2) ensures that we can model many systems, whereas item 3) characterizes how easy it is to model systems: the smaller a model is, the more readable it is.

4.5.1. *Diagonal clock constraints*

In the timed automata model we have presented, clock constraints that can be used on transitions are rather simple and can only compare the value of a clock with a constant. In [ALU 90, ALU 94a], another type of constraint was mentioned, the diagonal constraint, which allow tests of the form $x - y \bowtie c$ where x and y are clocks, $\bowtie$ is a comparison operator and c is an integer. The extended timed automata model using this kind of constraint satisfies the following properties:

– checking reachability properties is also a PSPACE-complete problem [ALU 94a];

– diagonal constraints do not add expressiveness to the original model [BÉR 98];

– diagonal constraints give conciseness to the model [BOU 05a].

The decidability of the reachability problem was already proved in the original papers [ALU 90, ALU 94a], and also relies on the construction of a region equivalence, which refines the one presented in section 4.3, and the complexity remains the same. The second property (which concerns the expressiveness) is well-known, and it has been proved in [BÉR 98]: it consists of removing one-by-one diagonal constraints, and of building an equivalent timed automaton without diagonal constraints (the equivalence is the strong timed bisimulation). The construction which removes one diagonal constraint is illustrated on Figure 4.4 (here, we remove the constraint $x - y \leqslant c$ where c is a non-negative integer). The key idea of the construction is that the truth value of the diagonal constraint $x - y \leqslant c$ remains unchanged when time elapses, and can only change when one of the two clocks x or y is reset to zero. Thus, we make two copies of the original automaton: in one of them the constraint $x - y \leqslant c$ will be satisfied, whereas in the other one, $x - y > c$ will hold. When x or y is reset to zero, we move from one copy to the other one, depending on the values of the clocks. For instance, if we reset clock y, we move to the copy where $x - y \leqslant c$ holds if $x \leqslant c$, and we move to the copy where $x - y > c$ holds if $x > c$. This construction doubles the size of the automaton, inducing an exponential blowup (in the number of diagonal constraints) for removing all diagonal constraints. This exponential blowup is unavoidable in general, as timed automata with diagonal constraints are exponentially more concise than classical timed automata [BOU 05a], which means that systems can be modeled using exponentially more succinct automata if diagonal constraints are used.

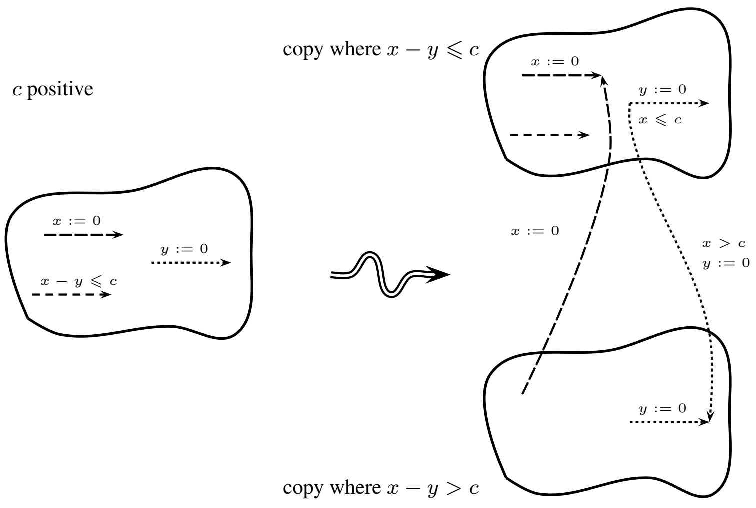


Figure 4.4. *Diagonal constraints are removed one-by-one*

4.5.2. *Additive clock constraints*

Other types of constraints can be added to the model of timed automata. We consider here the so-called additive clock constraints, i.e., constraints of the form $x + y \bowtie c$ where x and y are clocks, $\bowtie$ is a comparison operator and c is a positive integer. This extension has been studied in [BÉR 00], and it makes it possible to recognize timed languages which are not recognized by any classical timed automaton. The automaton in Figure 4.5 recognizes such a timed language: actions a are done at time points $\frac{1}{2}$, $\frac{3}{4}$, $\frac{7}{8}$, $\frac{15}{16}$, etc.

$$L^+ = \{(a^n, t_1 \cdots t_n) \mid n \geqslant 1 \text{ and } t_i = 1 - \tfrac{1}{2^i}\}$$

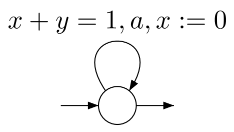


Figure 4.5. *A timed language not recognized by any classical timed automaton*

This model of timed automata with additive clock constraints satisfies the following properties [BÉR 00]:

– checking reachability properties in this extended model is decidable when restricting to automata with two clocks;

– checking reachability properties in this extended model is undecidable for automata with four clocks or more.

The decidability of the model with no more than two clocks also relies on the construction of a region equivalence, the set of regions being a refinement of the classical set of regions (see Figure 4.6). For models with four clocks or more, the model becomes undecidable. The proof is rather involved but also interesting, and uses the small automaton of Figure 4.5 several times.

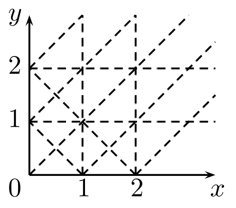


Figure 4.6. *Region equivalence for timed automata with additive clock constraints and two clocks*

Note that it is not known whether the reachability problem is decidable or not for timed automata with additive clock constraints and three clocks. However, it has been proved that a simple extension of the classical construction based on an equivalence of finite index cannot be used [ROB 04].

4.5.3. *Internal actions*

In finite automata, it is well-known that internal actions (also called ε-transitions in this context) can be removed and hence do not increase the expressiveness of finite automata (see for instance [HOP 79]). The timed automata framework is, maybe surprisingly, much different: though internal actions do not change anything to the decidability of reachability properties (the construction of the region automaton can be done similarly), they add expressive power to the model [BÉR 98]. The automaton depicted in Figure 4.7 recognizes a timed language that cannot be recognized by a classical timed automaton. This language is the set of timed words over a single letter a, where every a is done at an integral even date: at every two time units, either the transition labeled by a is taken, or the transition labeled by ε is taken.

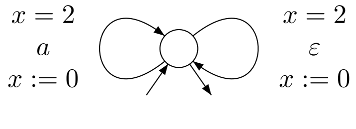


Figure 4.7. *A timed language not recognized by any classical timed automaton*

4.5.4. *Updates of clocks*

In the original model, there is only a single operation that can modify the value of the clocks (apart from time elapsing), which is the reset to zero. It is hence natural to consider more complicated operations on clocks. An update is an operation of the form $x :\bowtie c$ or $x :\bowtie y + c$, where x and y are clocks, $\bowtie$ is a comparison operator, and c is a constant. For instance, the update $x :\leqslant c$ means that we assign non-deterministically a value smaller than (or equal to) c to the clock x; the update $x := y - 1$ means that we assign to clock x the value of y decremented by 1. Classical reset to zero hence corresponds to updates of the form $x := 0$. Timed automata that use these updates, called updatable timed automata, have been studied in [BOU 04b]. It is fairly straightforward to check that the general model is undecidable as all these updates are rather powerful (it is possible to increment, decrement, test to zero). However, several subclasses have been proved decidable, and we summarize some of the most noticeable results of [BOU 04b]. The reachability problem is:

– decidable for timed automata with updates of the form $x := c$;

– decidable for timed automata with self-incrementation[3] but without diagonal constraints;

– undecidable for timed automata with self-incrementation and diagonal constraints;

– undecidable for timed automata with self-decrementation[4].

Once more, decidability results are consequences of a region automaton construction. The refinement of the region equivalence is illustrated in Figure 4.8 for timed automata with clock constraints $\{x - y < 1,\ y > 1\}$ and updates $\{x := 0,\ y := 0,\ y := 1\}$. The classical set of regions would be the set of regions depicted with dashed lines, but it is not correct in that wider framework (the image of the gray region by update $y := 1$ overlaps two regions and does not satisfy a time-abstract bisimulation property); it is thus necessary to refine and add the dotted line to distinguish $x = 2$.

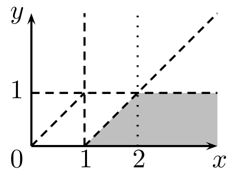


Figure 4.8. *A set of regions for automata using constraints* $\{x - y < 1, y > 1\}$ *and updates* $\{x := 0, y := 0, y := 1\}$

3. I.e., with updates of the form $x := x + 1$.

4. I.e., with updates of the form $x := x - 1$.

The reachability problem is undecidable for timed automata using self-decrementation. Indeed, we can easily encode the behavior of a two-counter machine with these automata: the value of counter C is stored in clock x_c, incrementing this counter is encoded by letting one time unit elapse, and then by decrementing the clock associated with the second counter by 1; decrementing this counter is directly encoded using the update $x_c := x_c - 1$.

Note that classes of updatable timed automata that have been proved decidable can be transformed into equivalent classical timed automata with internal actions [BOU 04b][5]. On the other hand, these updatable timed automata are exponentially more concise than classical timed automata [BOU 05a].

Finally, it is worth noticing that updates of clocks are a kind of macros which are very useful to model real systems. For instance, we can mention the modelization scheduling problems which naturally uses updates [FER 02].

4.5.5. *Linear hybrid automata*

Linear hybrid automata extend timed automata in several directions:

– general linear constraints can be used, for instance constraints of the form

$$3x_1 + 4x_2 - 2x_3 < 56;$$

– very rich updates can be used, for instance, affine functions on variables;

– derivatives of variables can change from one location to the other: instead of having only clocks (whose derivative is always 1), we can use dynamical variables.

In fact, even just one of these extensions lead to the undecidability of all verification questions! We have already mentioned that for additive clock constraint, and updates of clocks. It is also the case for variables with several possible slopes in the model: the reachability problem is undecidable for timed automata in which a single variable can have two different slopes. We refer to [HEN 98] or more recently to [RAS 05] for surveys on these questions, where some of the decidable classes of linear hybrid automata are described.

Note that looking for decidable subclasses of hybrid automata is an important research topic (for instance, rectangular initialized hybrid automata are decidable [HEN 98], and o-minimal hybrid automata are decidable [LAF 00]). It is also important to find either semi-algorithms,[6] or approximation and optimization algorithms

5. The equivalence relation is then the timed language equivalence.

6. That is, computation procedures that may not terminate. A semi-algorithm can then answer either "The property is satisfied", "The property is not satisfied", or "I don't know".

for undecidable classes of hybrid automata. Indeed, undecidable classes can have a great interest in practice, like the class of p-automata [BÉR 99] for the description of telecommunication protocols, or stopwatch automata (i.e., timed automata in which clocks can be stopped for a while) for scheduling problems with pre-emption.

Most of these methods rely on the manipulation of linear constraints to represent a set of states of the system [ALU 95], and algorithms that use polyhedra libraries (as the Parma Polyhedra Library[7]). One of the most prominent tools for linear hybrid systems is HyTech, which allows both the step-by-step computation of successors (or predecessors) of sets of states, and fix-point computations (although without a guarantee that the computation will terminate). See [HEN 97] for more details and examples. The tool HyTech can be downloaded at `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`.

4.6. Subclasses of timed automata

As explained above, timed automata are a very expressive formalism and almost every extension leads to undecidability of verification problems. Instead of extending the expressiveness of timed automata, it is also possible to consider restricted versions in order to obtain new properties, for instance efficient algorithms. In this section, we present some of the classical restricted classes: the event-recording automata, the one-clock timed automata and the timed extensions of classical Kripke structures.

4.6.1. *Event-recording automata*

In this subclass, the set of automata clocks is $X_\Sigma = \{x_a \mid a \in \Sigma\}$: every action has a corresponding clock and every clock is associated with an action. The definition of the event-recording automata (see [ALU 94b]) also requires that clock x_a is reset to zero when an a-transition is performed. Then, given a configuration (q, v), the value $v(x_a)$ is the time elapsed since the last occurrence of an a action[8].

The event-recording automata (ER-TA) have important properties. First, non-determinism can be removed: any ER-TA can be transformed into a deterministic ER-TA. Secondly, the timed languages associated with this class are closed under complement. However, note that from the complexity point of view, there is no change: emptiness checking remains PSPACE-complete (the same holds for TCTL model checking).

7. See `http://www.cs.unipr.it/ppl/`.

8. A variant – the *event-predicting* automata – consists of storing in $v(x_a)$ the amount of time *before* the next action a (in this case, clocks are initialized with a negative value).

4.6.2. *One-clock timed automata*

In [COU 92], it is shown that reachability of a control location in timed automata with three clocks is PSPACE-hard. This has prompted the study of model checking for timed automata with one or two clocks.

In [LAR 04], the following results have been proved for the one-clock timed automata (1C-TA for short):

– reachability of a control location in 1C-TA is NLOGSPACE-complete (i.e. the same complexity class as reachability in standard graphs);

– there exist polynomial time algorithms for model checking $\mathsf{TCTL}_{\leqslant,\geqslant}$ over 1C-TA. ($\mathsf{TCTL}_{\leqslant,\geqslant}$ is the fragment of TCTL where timing constraints "$= c$" are forbidden);

– model checking TCTL over 1C-TA is PSPACE-complete.

The main result is that model checking 1C-TA can be done efficiently if the specification is stated with $\mathsf{TCTL}_{\leqslant,\geqslant}$. Note that the complexity blow-up induced by the *punctuality* (the constraint "$= c$") occurs in other cases (for instance, in the case of linear-time timed logics [ALU 96]).

The model checking algorithm for $\mathsf{TCTL}_{\leqslant,\geqslant}$ over 1C-TA works as follows. Given a 1C-TA $\mathcal{A}$ and a $\mathsf{TCTL}_{\leqslant,\geqslant}$ formula Φ, we compute, for any state q and any subformula ψ, the set of valuations v such that $(q, v) \models \psi$. As $\mathcal{A}$ has only one clock, such a valuation v is a unique value, and the sets of valuations $\mathsf{Sat}[q, \psi]$ can be represented as a union of intervals $\bigcup_i \langle \alpha_i, \beta_i \rangle$, with $\langle \in \{[, (\}, \rangle \in \{],)\}$ and $\alpha_i, \beta_i \in \mathbb{N} \cup \{\infty\}$. We can show that the number of intervals in $\mathsf{Sat}[\ell, \varphi]$ is bounded by $2 \cdot |\varphi| \cdot |\mathcal{A}|$.

For example, consider the subformula $\mathsf{E}\phi\mathsf{U}_{\leqslant c}\psi$ and assume that the sets $\mathsf{Sat}[q, \varphi]$ and $\mathsf{Sat}[q, \psi]$ have already been computed for any q. The aim is to compute the minimal duration – denoted $\delta^{\min}(q, v)$ – to reach from (q, v) a ψ state along a path satisfying φ. To compute the function $\delta^{\min}$, we first build a simplified region automaton (its size is polynomial in $|\mathcal{A}|$ and $|\Phi|$) whose states are pairs (q, γ), where γ is an interval of valuations such that the truth values of φ, ψ and any guard in $\mathcal{A}$ do not change along γ. This property entails that the function $\delta^{\min}$ has a special form over every γ: it is either decreasing with slope -1 (the shortest paths to reach ψ go through the rightmost position of γ), or constant (the shortest paths to reach ψ have to perform an action transition with a reset of the clock before any delay transition) or it combines the two previous cases, that is, it is first constant over an subinterval of γ and then decreasing. Thus, every function $\delta^{\min}_{|(q,\gamma)}$ can be defined by its value on the leftmost and rightmost positions of γ, and these value can be easily computed with a shortest path algorithm. It remains to use the threshold $\leqslant c$ to find the intervals where $\mathsf{E}\phi\mathsf{U}_{\leqslant c}\psi$ is true [LAR 04].

Note also that one-clock timed automata have other very interesting properties:

– the timed language inclusion is decidable for finite words for 1C-TA [OUA 04] (but remains undecidable for infinite words [ABD 05]);

– checking emptiness is decidable for one-clock *alternating* timed automata [LAS 05, OUA 05];

– model checking one-clock *probabilistic* timed automata can be done in polynomial time for $\mathsf{PTCTL}_{\leqslant,\geqslant}$ formulae, and almost sure reachability is P-complete [JUR 07];

– model checking WCTL is decidable (in PSPACE) for the *priced* timed automata with one clock [BOU 07a]: WCTL is a specification language (its syntax is the same as TCTL) to express quantitative properties over the cost of executions in priced timed automata (where a cost slope is associated with every location);

– computing optimal costs can be done for *priced* timed *game* with one clock [BOU 06].

Note that these properties no longer hold when the timed automata have two clocks (reachability is NP-hard for two-clocks timed automata [LAR 04], almost-sure reachability is EXPTIME-complete in two clocks probabilistic timed automata [JUR 07], etc.).

4.6.3. *Discrete-time models*

Instead of considering $\mathbf{R}$, it is possible to use a discrete-time domain. For example, we can consider the semantics of timed automata with integral clocks. This change does not modify the main complexity results of model checking: for example, reachability and the TCTL model checking remain PSPACE-complete. To obtain polynomial time algorithms, we need to consider simpler models.

In many works, classical Kripke structures have been used to model real-time systems with the hypothesis that every transition takes exactly one time unit. In this case, we can use TCTL formulae to specify quantitative properties (over the number of transitions) along the paths. With this simple approach, there exist polynomial time model checking algorithms for model checking TCTL [EME 92], and they can also be extended to models where the transitions take 0 or 1 time unit [LAR 03].

A natural extension consists of associating integral durations with the transitions of a Kripke structure. Several semantics can be defined for these systems. In this framework, the main interesting result is that model checking $\mathsf{TCTL}_{\leqslant,\geqslant}$ can be done in polynomial time (contrary to TCTL, whose model checking is either Δ_2^p-complete or PSPACE-complete depending on the semantics which is chosen) [LAR 06]. This polynomial-time algorithm has been implemented in the tool TSMV [MAR 04].

4.7. Algorithms for timed verification

In this section, we describe algorithms implemented in tools such as Uppaal or Kronos for verifying timed automata. Indeed, in practice the region automaton construction is not used in tools because the region partition is too refined and hence it is not efficient to manipulate the regions. Tools should use the symbolic representation called *zones*, and rely on on-the-fly algorithms.

There are mainly two families of (semi-)algorithms for analyzing reachability properties of systems. The first, called forward analysis, consists of computing iteratively the successors of the initial states and of checking that the state we want to reach is eventually computed or not. The second, called backward analysis, consists of computing iteratively the predecessors of the states we want to reach and of checking that an initial state is eventually computed or not. These methods are generic and are used in many contexts, for instance on the model of linear hybrid automata that we already mentioned in section 4.5.

Before presenting these analysis methods, we first present the most commonly used symbolic representation for the verification of timed systems.

4.7.1. *A symbolic representation for timed automata: the zones*

The set of configurations of a timed automaton is infinite. To verify this model, it is thus mandatory to be able to manipulate large sets of configurations and thus to have an efficient symbolic representation for these sets of states. The most commonly used is the zone representation: a zone is a set of valuations defined by a conjunction of atomic constraints of the form $x \bowtie c$ or $x - y \bowtie c$ where x and y are clocks, $\bowtie$ is a comparison operator, and c is a constant. Thus, in the forward and backward analysis algorithms, objects that are manipulated are pairs (ℓ, Z) where ℓ is a location and Z a zone.

Many operations can be performed using this representation:

– the future of a zone Z, defined by $\overrightarrow{Z} = \{v + t \mid v \in Z \text{ and } t \in \mathbb{T}\}$;

– the past of a zone Z, defined by $\overleftarrow{Z} = \{v - t \mid v \in Z \text{ and } t \in \mathbb{T}\}$;

– the intersection of Z and Z', defined by $Z \cap Z' = \{v \mid v \in Z \text{ and } v \in Z'\}$;

– the reset to zero $r \subseteq X$ of Z, defined by $[r \leftarrow 0]Z = \{[r \leftarrow 0]v \mid v \in Z\}$;

– the relaxation with respect to $r \subseteq X$ of Z, defined by $[r \leftarrow 0]^{-1}Z = \{v \mid [r \leftarrow 0]v \in Z\}$.

These operations, defined as first order formulae over zones, preserve zones (see the Fourier-Motzkin elimination principle [SCH 98]).

We now present the backward analysis algorithm as it is the simplest one. We will then turn to the forward analysis algorithm, which is indeed the most commonly used method, but also the most complicated one.

4.7.2. *Backward analysis in timed automata*

As already said, the backward analysis consists of computing step-by-step the predecessors of the final configurations, starting with the one step predecessors, then the two steps predecessors, etc. and of checking whether an initial state is eventually computed. If such an initial state is computed, it means that the goal location is reachable, and if such an initial state is not computed, it means that the goal location is not reachable. The principle of the backward analysis is illustrated in Figure 4.9.

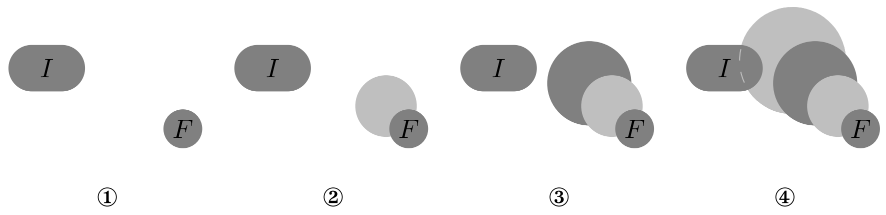


Figure 4.9. *Backward analysis: step-by-step, predecessors of goal states are computed*

One step of the backward analysis can easily be computed using zones. Indeed, if $t = \ell \xrightarrow{g,a,r} \ell'$ is a transition of the automaton and if Z' is a zone, the set of one-step predecessors of (ℓ', Z') when taking transition t is the set of configurations (ℓ, v) where v is in the zone $Z = \overleftarrow{g \cap [r \leftarrow 0]^{-1}(Z' \cap (r = 0))}$.

The characteristic of the backward analysis is that the iterative computation always terminates: indeed it can be proved that if Z' is a zone and if this zone is a union of regions (see section 4.3), then the zone Z' we have described before is a zone and also a union of regions! As there are finitely many regions, there are only finitely many pairs (ℓ, Z) which can be computed.

Though the backward analysis has some non-negligible qualities, in practice, it is not commonly implemented in tools, and forward analysis is preferred. There are numerous reasons for this implementation choice: forward analysis only visits reachable states, *i.e.*, states that are relevant in the system; furthermore, backward analysis is not appropriate for verifying systems defined with high-level data structures such as integral variables or C-like instructions, etc. For instance, the Uppaal tool (see section 4.8) only implements the forward analysis paradigm.

4.7.3. *Forward analysis of timed automata*

As previously stated, forward analysis consists of computing step-by-step the successors of the initial configurations, starting with the one step successors, then the two steps successors, etc. and of checking whether a goal location is eventually computed. If such a final location is computed, it means that the goal location is reachable, and if such an initial state is not computed, it means that the goal location is not reachable. The principle of the forward analysis is illustrated in Figure 4.10.

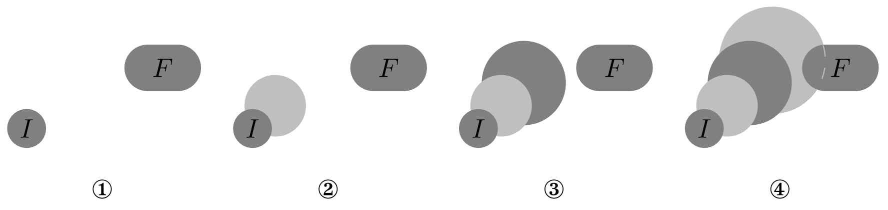


Figure 4.10. *Forward analysis: step-by-step, successors of initial states are computed*

One step of the forward analysis algorithm can be computed using zones. Indeed, if $t = \ell \xrightarrow{g,a,r} \ell'$ is a transition of the timed automaton and if Z is a zone, the set of successors in one step of (ℓ, Z) by taking transition t is the set of states (ℓ', v') where v' belongs to the zone $Z' = [r \leftarrow 0](g \cap \overrightarrow{Z})$.

Contrary to the backward computation, the iterative forward computation does not terminate in general. This is illustrated by the timed automaton of Figure 4.11. In this example, each iteration of the algorithm increases the value of the clock by 1, and the computation will thus never terminate.

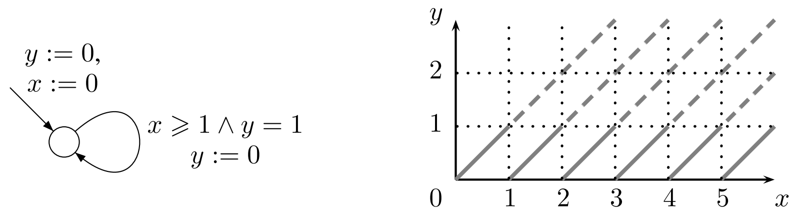


Figure 4.11. *The iterative forward computation may not terminate*

To overcome this termination problem, an abstraction operator is usually applied at each iteration of the computation. In other works, this abstraction operator is called normalization, or extrapolation; we use the latter formulation here. We assume that k is the largest constant appearing in the constraints of the timed automaton. If Z is a zone, the extrapolation of Z with respect to k is the smallest zone which contains Z and which is defined with constants between $-k$ and $+k$. The idea behind this operator is the following: the automaton cannot distinguish between clock values above k, and it may thus not be relevant to keep in the zone information above k. It is worth noticing first that applying this extrapolation at each iteration of the algorithm ensures termination of the computation as there are only finitely many zones defined with constants between $-k$ and $+k$. However, there is another problem: at each iteration, an over-approximation of the set of states which is actually reachable is computed. It may thus happen that a location is computed whereas it is not reachable.

In [BOU 04a], it is proven that this iterative and abstracted forward computation is correct for the class of timed automata without diagonal constraints, but incorrect for the class of timed automata with diagonal constraints.

4.7.4. *A data structure for timed systems: DBMs*

The DBM acronym stands for *difference bound matrix*. It is a rather classical data structure used for representing systems of difference constraints [COR 90], and they have in particular a great interest for the verification of timed systems because they can be used to represent zones. DBMs were first used to analyze time Petri nets [BER 83], and they are now intensively used to analyze timed automata [DIL 90].

If n is the number of clocks, a DBM M is an $(n+1)$-square matrix with integral coefficients[9]. If $M = (m_{i,j})_{0 \leqslant i,j \leqslant n}$, the coefficient $m_{i,j}$ represents the constraint $x_i - x_j \leqslant m_{i,j}$ where $\{x_i \mid 1 \leqslant i \leqslant n\}$ is the set of clocks and x_0 is a fictitious clock whose value is always 0. Thus, to represent a constraint $x_i \leqslant 6$, we will write $m_{i,0} = 6$ as this constraint is equivalent to $x_i - x_0 \leqslant 6$.

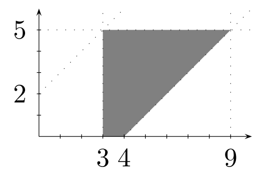


Figure 4.12. *Zone defined by the constraint* $(x_1 \geqslant 3) \wedge (x_2 \leqslant 5) \wedge (x_1 - x_2 \leqslant 4)$

The following DBM represents the set of valuations defined by the constraint $(x_1 \geqslant 3) \wedge (x_2 \leqslant 5) \wedge (x_1 - x_2 \leqslant 4)$, and is represented in Figure 4.12:

$$\begin{array}{cc} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} +\infty & -3 & +\infty \\ +\infty & +\infty & 4 \\ 5 & +\infty & +\infty \end{pmatrix} \end{array}.$$

9. In general, coefficients need to be pairs $(m, \prec)$ where m is an integer and $\prec$ is either $<$, or $\leqslant$, but here, to simplify the presentation, we forget about comparison operators in DBMs.

A coefficient $+\infty$ means that there is no constraint on the corresponding clock difference. The coefficient $m_{0,1} = -3$ represents the constraints $x_1 \geqslant 3$ because this constraint is equivalent to $x_0 - x_i \leqslant -3$.

Every DBM represents a zone, and every zone can be represented by a DBM. However, a zone can be represented by several DBMs (for instance, in the previous DBM, if we replace the coefficient $m_{1,0} = +\infty$ by $m_{1,0} = 9$, it will not change the zone which is represented). There exists a normal form for DBMs, which can be computed using the Floyd-Warshall shortest paths algorithm [COR 90]: the DBM which is obtained stores the strongest constraints which define the corresponding zone. For the previous example, the normal form DBM is:

$$\begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix}.$$

All operations that are needed for both the backward and the forward analysis iterative computations can be done using DBMs (see [CLA 99, BOU 04a] for a detailed description of operations using DBMs).

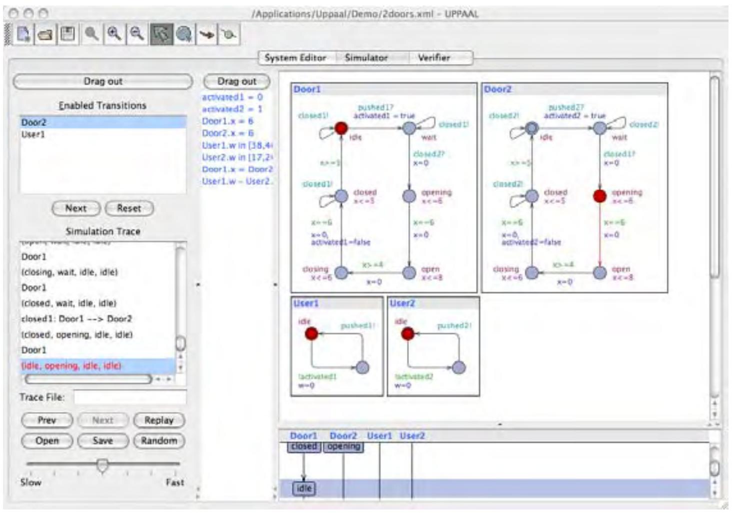
DBMs are basic data structures for manipulating sets of configurations of timed automata, but several improvements can be made, for instance, a minimization of DBMs [LAR 97a] or the use of CDDs (*clock difference diagrams*) [LAR 99] or more recently of federations [DAV 06], which makes it possible to represent and manipulate more compactly the unions of DBMs.

4.8. The model-checking tool Uppaal

Uppaal is a model-checking tool for verifying timed systems. It has been jointly developed by Uppsala University (Sweden) and Aalborg University (Denmark) [LAR 97b]. (This tool can be downloaded at `http://www.uppaal.com/`.) The model that can be verified by Uppaal is a variant of the classical timed automata model. This model is syntactically very rich as we can explicitly add urgency in the model (for instance, we can enforce a transition to be taken immediately, without any delay), we can enforce atomicity of several transitions (a sequence of transitions must then be taken instantaneously), we can add C-like instructions, etc. All these features do not add expressivity to the model, but they make the modeling phase easier, as we can build rather concise and readable models.

Properties that can be verified using the Uppaal tool are reachability properties, safety properties, and response properties. A tutorial for this tool is available online [BEH 04].

Apart from the modeling GUI and the verification module, Uppaal has a simulation module in which it is possible to “play” with the model and hence have a first



```
Door2
User1
```

```
activated1 = 0
activated2 = 1
Door1.x = 6
Door2.x = 6
User1.w in [38,4
User2.w in [17,2
Door1.x = Door2
User1.w - User2.
```

```
Door1
(closing, wait, idle, idle)
Door1
(closed, wait, idle, idle)
closed1: Door1 --> Door2
(closed, opening, idle, idle)
Door1
(idle, opening, idle, idle)
```

Figure 4.13. *The* Uppaal *tool*

check that the model does what it is expected to do. A screenshot of the tool is given in Figure 4.13.

The Uppaal tool has been developed for more than ten years, and it has been successfully used to verify industrial systems. For instance, we can mention audio protocols like [BEN 02], or the Bang & Olufsen protocol whose analysis has located a known bug, and for which a validated correction has been provided [HAV 97].

The current version of Uppaal is 4.0 and the new features are described in [BEH 06].

4.9. Bibliography

[ABD 05] ABDULLA P. A., DENEUX J., OUAKNINE J., WORRELL J., "Decidability and complexity results for timed automata via channel machines", *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, vol. 3580 of *Lecture Notes in Computer Science*, Springer, p. 1089–1101, 2005.

[ACE 98] ACETO L., BURGUEÑO A., LARSEN K. G., "Model-checking via reachability testing for timed automata", *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, vol. 1384 of *Lecture Notes in Computer Science*, Springer, p. 263–280, 1998.

[ACE 02] ACETO L., LAROUSSINIE F., "Is your model-checker on time? On the complexity of model-checking for timed modal logics", *Journal of Logic and Algebraic Programming*, vol. 52–53, p. 7–51, 2002.

[ACE 03] ACETO L., BOUYER P., BURGUEÑO A., LARSEN K. G., "The power of reachability testing for timed automata", *Theoretical Computer Science*, vol. 300, num. 1–3, p. 411–475, 2003.

[ALU 90] ALUR R., DILL D., "Automata for modeling real-time systems", *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, vol. 443 of *Lecture Notes in Computer Science*, Springer, p. 322–335, 1990.

[ALU 93a] ALUR R., COURCOUBETIS C., DILL D., "Model-checking in dense real-time", *Information and Computation*, vol. 104, num. 1, p. 2–34, 1993.

[ALU 93b] ALUR R., HENZINGER TH. A., "Real-time logics: complexity and expressiveness", *Information and Computation*, vol. 104, num. 1, p. 35–77, 1993.

[ALU 94a] ALUR R., DILL D., "A theory of timed automata", *Theoretical Computer Science*, vol. 126, num. 2, p. 183–235, 1994.

[ALU 94b] ALUR R., FIX L., HENZINGER TH. A., "A determinizable class of timed automata", *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, vol. 818 of *Lecture Notes in Computer Science*, Springer, p. 1–13, 1994.

[ALU 94c] ALUR R., HENZINGER TH. A., "A really temporal logic", *Journal of the ACM*, vol. 41, num. 1, p. 181–204, 1994.

[ALU 95] ALUR R., COURCOUBETIS C., HALBWACHS N., HENZINGER TH. A., HO P.-H., NICOLLIN X., OLIVERO A., SIFAKIS J., YOVINE S., "The algorithmic analysis of hybrid systems", *Theoretical Computer Science*, vol. 138, num. 1, p. 3–34, 1995.

[ALU 96] ALUR R., FEDER T., HENZINGER TH. A., "The benefits of relaxing punctuality", *Journal of the ACM*, vol. 43, num. 1, p. 116–146, 1996.

[BEH 04] BEHRMANN G., DAVID A., LARSEN K. G., "A tutorial on Uppaal", *Proc. 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real-Time (SFM-04:RT)*, vol. 3185 of *Lecture Notes in Computer Science*, Springer, p. 200–236, 2004.

[BEH 06] BEHRMANN G., DAVID A., LARSEN K. G., HÅKANSSON J., PETTERSSON P., YI W., HENDRIKS M., "Uppaal 4.0", *Proc. 3rd International Conference on the Quantitative Evaluation of Systems (QEST'06)*, IEEE Computer Society Press, p. 125–126, 2006.

[BEN 02] BENGTSSON J., GRIFFIOEN W. D., KRISTOFFERSEN K. J., LARSEN K. G., LARSSON F., PETTERSSON P., YI W., "Automated verification of an audio-control protocol using Uppaal", *Journal of Logic and Algebraic Programming*, vol. 52–53, p. 163–181, 2002.

[BER 83] BERTHOMIEU B., MENASCHE M., "An enumerative approach for analyzing time Petri nets", *Proc. IFIP 9th World Computer Congress*, vol. 83 of *Information Processing*, North-Holland/ IFIP, p. 41–46, 1983.

[BÉR 98] BÉRARD B., DIEKERT V., GASTIN P., PETIT A., "Characterization of the expressive power of silent transitions in timed automata", *Fundamenta Informaticae*, vol. 36, num. 2–3, p. 145–182, 1998.

[BÉR 99] BÉRARD B., FRIBOURG L., "Automated verification of a parametric real-time program: the ABR Conformance Protocol", *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, vol. 1633 of *Lecture Notes in Computer Science*, Springer, p. 96–107, 1999.

[BÉR 00] BÉRARD B., DUFOURD C., "Timed automata and additive clock constraints", *Information Processing Letters*, vol. 75, num. 1–2, p. 1–7, 2000.

[BOU 04a] BOUYER P., "Forward Analysis of Updatable Timed Automata", *Formal Methods in System Design*, vol. 24, num. 3, p. 281–320, 2004.

[BOU 04b] BOUYER P., DUFOURD C., FLEURY E., PETIT A., "Updatable timed automata", *Theoretical Computer Science*, vol. 321, num. 2–3, p. 291–345, 2004.

[BOU 05a] BOUYER P., CHEVALIER F., "On conciseness of extensions of timed automata", *Journal of Automata, Languages and Combinatorics*, vol. 10, num. 4, p. 393–405, 2005.

[BOU 05b] BOUYER P., CHEVALIER F., MARKEY N., "On the expressiveness of TPTL and MTL", *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, vol. 3821 of *Lecture Notes in Computer Science*, Springer, p. 432–443, 2005.

[BOU 06] BOUYER P., LARSEN K. G., MARKEY N., RASMUSSEN J. I., "Almost optimal strategies in one-clock priced timed automata", *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'06)*, vol. 4337 of *Lecture Notes in Computer Science*, Springer, p. 345–356, 2006.

[BOU 07a] BOUYER P., LARSEN K. G., MARKEY N., "Model-checking one-clock priced timed automata", *Proc. 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*, vol. 4423 of *Lecture Notes in Computer Science*, Springer, p. 108–122, 2007.

[BOU 07b] BOUYER P., MARKEY N., OUAKNINE J., WORRELL J., "The cost of punctuality", *Proc. 21st Annual Symposium on Logic in Computer Science (LICS'07)*, IEEE Computer Society Press, p. 109–118, 2007.

[CLA 81] CLARKE E. M., EMERSON E. A., "Design and synthesis of synchronous skeletons using branching-time temporal logic", *Proc. 3rd Workshop on Logics of Programs (LOP'81)*, vol. 131 of *Lecture Notes in Computer Science*, Springer Verlag, p. 52–71, 1981.

[CLA 99] CLARKE E., GRUMBERG O., PELED D., *Model checking*, The MIT Press, Cambridge, Massachusetts, 1999.

[COR 90] CORMEN TH. H., LEISERSON C. E., RIVEST R. L., *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1990.

[COU 92] COURCOUBETIS C., YANNAKAKIS M., "Minimum and maximum delay problems in real-time systems", *Formal Methods in System Design*, vol. 1, num. 4, p. 385–415, 1992.

[DAV 06] DAVID A., "Merging DBMs efficiently", *Proc. 17th Nordic Workshop on Programming Theory*, DIKU, University of Copenhagen, p. 54–56, 2006.

[DIL 90] DILL D., "Timing assumptions and verification of finite-state concurrent systems", *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems (1989)*, vol. 407 of *Lecture Notes in Computer Science*, Springer, p. 197–212, 1990.

[EME 91] EMERSON E. A., *Temporal and Modal Logic*, vol. B (Formal Models and Semantics) of *Handbook of Theoretical Computer Science*, p. 995–1072, MIT Press Cambridge, 1991.

[EME 92] EMERSON E. A., MOK A. K., SISTLA A. P., SRINIVASAN J., "Quantitative temporal reasoning", *Real-Time Systems*, vol. 4, num. 4, p. 331–352, 1992.

[FER 02] FERSMAN E., PETTERSON P., YI W., "Timed automata with asynchronous processes: schedulability and decidability", *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, vol. 2280 of *Lecture Notes in Computer Science*, Springer, p. 67–82, 2002.

[HAV 97] HAVELUND K., SKOU A., LARSEN K. G., LUND K., "Formal modeling and analysis of an audio/video protocol: an industrial case study using Uppaal", *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, IEEE Computer Society Press, p. 2–13, 1997.

[HEN 85] HENNESSY M., MILNER R., "Algebraic laws for nondeterminism and concurrency", *Journal of the ACM*, vol. 32, num. 1, p. 137–161, 1985.

[HEN 94] HENZINGER TH. A., NICOLLIN X., SIFAKIS J., YOVINE S., "Symbolic model-checking for real-time systems", *Information and Computation*, vol. 111, num. 2, p. 193–244, 1994.

[HEN 97] HENZINGER TH. A., HO P.-H., WONG-TOI H., "HyTech: A model-checker for hybrid systems", *Journal on Software Tools for Technology Transfer*, vol. 1, num. 1–2, p. 110–122, 1997.

[HEN 98] HENZINGER TH. A., KOPKE P. W., PURI A., VARAIYA P., "What's decidable about hybrid automata?", *Journal of Computer and System Sciences*, vol. 57, num. 1, p. 94–124, 1998.

[HOP 79] HOPCROFT J. E., ULLMAN J. D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[JUR 07] JURDZIŃSKI M., LAROUSSINIE F., SPROSTON J., "Model checking probabilistic timed automata with one or two clocks", *Proc. of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'07)*, vol. 4424 of *Lecture Notes in Computer Science*, Springer, p. 170–184, 2007.

[KOY 90] KOYMANS R., "Specifying real-time properties with metric temporal logic", *Real-Time Systems*, vol. 2, num. 4, p. 255–299, 1990.

[LAF 00] LAFFERRIERE G., PAPPAS G. J., SASTRY S., "O-minimal hybrid systems", *Mathematics of Control, Signals, and Systems*, vol. 13, num. 1, p. 1–21, 2000.

[LAR 90] LARSEN K. G., "Proof systems for satisfiability in Hennessy-Milner logic with recursion", *Theoretical Computer Science*, vol. 72, num. 2–3, p. 265–288, 1990.

[LAR 95a] LAROUSSINIE F., LARSEN K. G., WEISE C., "From timed automata to logic – and back", *Proc. 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, vol. 969 of *Lecture Notes in Computer Science*, Springer, p. 529–539, 1995.

[LAR 95b] LARSEN K. G., PETTERSSON P., YI W., "Model-checking for real-time systems", *Proc. 10th International Conference on Fundamentals of Computation Theory (FCT'95)*, vol. 965 of *Lecture Notes in Computer Science*, Springer, p. 62–88, 1995.

[LAR 97a] LARSEN K. G., LARSSON F., PETTERSSON P., YI W., "Efficient verification of real-time systems: compact data structure and state-space reduction", *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, IEEE Computer Society Press, p. 14–24, 1997.

[LAR 97b] LARSEN K. G., PETTERSSON P., YI W., "UPPAAL in a nutshell", *Journal of Software Tools for Technology Transfer*, vol. 1, num. 1–2, p. 134–152, 1997.

[LAR 99] LARSEN K. G., PEARSON J., WEISE C., YI W., "Clock difference diagrams", *Nordic Journal of Computing*, vol. 6, num. 3, p. 271–298, 1999.

[LAR 00] LAROUSSINIE F., SCHNOEBELEN PH., "The state-explosion problem from trace to bisimulation equivalence", *Proc. 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'00)*, vol. 1784 of *Lecture Notes in Computer Science*, Springer, p. 192–207, 2000.

[LAR 03] LAROUSSINIE F., SCHNOEBELEN PH., TURUANI M., "On the expressivity and complexity of quantitative branching-time temporal logics", *Theoretical Computer Science*, vol. 297, num. 1, p. 297–315, 2003.

[LAR 04] LAROUSSINIE F., MARKEY N., SCHNOEBELEN PH., "Model checking timed automata with one or two clocks", *Proc. 15th International Conference on Concurrency Theory (CONCUR'04)*, vol. 3170 of *Lecture Notes in Computer Science*, Springer, p. 387–401, 2004.

[LAR 06] LAROUSSINIE F., MARKEY N., SCHNOEBELEN PH., "Efficient timed model checking for discrete-time systems", *Theoretical Computer Science*, vol. 353, num. 1–3, p. 249–271, 2006.

[LAS 05] LASOTA S., WALUKIEWICZ I., "Alternating timed automata", *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, vol. 3441 of *Lecture Notes in Computer Science*, Springer, p. 250–265, 2005.

[MAR 04] MARKEY N., SCHNOEBELEN PH., "Symbolic model checking of simply-timed systems", *Proc. Joint Conference on Formal Modeling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, vol. 3253 of *Lecture Notes in Computer Science*, Springer, p. 102–117, 2004.

[OUA 04] OUAKNINE J., WORRELL J., "On the language inclusion problem for timed automata: closing a decidability gap", *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'04)*, IEEE Computer Society Press, p. 54–63, 2004.

[OUA 05] OUAKNINE J., WORRELL J., "On the decidability of metric temporal logic", *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05)*, IEEE Computer Society Press, p. 188–197, 2005.

[OUA 06] OUAKNINE J., WORRELL J., "On metric temporal logic and faulty turing machines", *Proc. 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, vol. 3921 of *Lecture Notes in Computer Science*, Springer, p. 217–230, 2006.

[PNU 77] PNUELI A., “The temporal logic of programs”, *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS’77)*, IEEE Computer Society Press, p. 46–57, 1977.

[PNU 81] PNUELI A., “The temporal semantics of concurrent programs”, *Theoretical Computer Science*, vol. 13, num. 1, p. 45–60, 1981.

[RAS 05] RASKIN J.-F., “An introduction to hybrid automata”, chapter in *Handbook of Networked and Embedded Control Systems*, p. 491–518, Springer, 2005.

[ROB 04] ROBIN A., “Aux frontières de la décidabilité...”, Master’s thesis, DEA Algorithmique, Paris, 2004.

[SCH 98] SCHRIJVER A., *Theory of Linear and Integer Programming*, Interscience Series in Discrete Mathematics and Optimization, Wiley, 1998.

[SCH 01] SCHNOEBELEN P., BÉRARD B., BIDOIT M., LAROUSSINIE F., PETIT A., *Systems and Software Verification – Model Checking Techniques and Tools*, Springer, 2001.

[STI 01] STIRLING C., *Modal and Temporal Properties of Processes*, Texts in Computer Science, Springer, 2001.

[TRI 98] TRIPAKIS S., YOVINE S., “Verification of the fast reservation protocol with delayed transmission using the tool Kronos”, *Proc. 4th IEEE Real-Time Technology and Applications Symposium (RTAS’98)*, IEEE Computer Society Press, p. 165–170, 1998.

[YI 90] YI W., “Real-time behavior of asynchronous agents”, *Proc. 1st International Conference on Theory of Concurrency (CONCUR’90)*, vol. 458 of *Lecture Notes in Computer Science*, Springer, p. 502–520, 1990.

[YOV 97] YOVINE S., “Kronos: A verification tool for real-time systems”, *Journal of Software Tools for Technology Transfer*, vol. 1, num. 1–2, p. 123–133, 1997.