Chapter 9

Verification of Timed Systems

9.1. Introduction

This chapter is devoted to the presentation of verification techniques for systems involving quantitative time constraints. Different ways of modeling such systems have been presented in Chapter 4. In this chapter, we present different results ranging from well established techniques (such as the region graph construction for timed automata) to more advanced issues (such as the implementability of timed automata).

Main methods to tame timed systems

Timed systems are a particular type of infinite-state systems in which the state space involves dense time variables. A general introduction on the verification techniques used to infinite-state systems can be found in Chapter 8. The most standard tool used to analyze timed systems is the symbolic representation of configurations, i.e. the identification of a framework that enables the finite representation of infinitely many configurations.

For some classes of timed systems, we can even use such a symbolic representation to transform the timed system into a finite-state system on which the property can be decided. In this chapter, we present such symbolic representations for several classes of timed systems. The first presentations of such constructions are the *state-class graph* for time Petri nets [BER 83], and the *region graph* for timed automata [ALU 94].

Chapter written by Pierre-Alain REYNIER.

These reductions to finite-state systems are usually based on an equivalence relation between configurations that admits a finite number of equivalence classes. However, it can also be the case that this equivalence admits an infinite number of classes, and still yields decidability results, for instance if there exists a well quasi-order on the equivalence classes.

From decidability to efficient procedures

Although the previously mentioned techniques allow decidability results to be proved, the algorithms resulting from them are not always the most efficient. For instance, for the class of timed automata, while the region graph construction is comonly used to prove decidability, it is never computed in practice. As an alternative, the symbolic representation of *zones* is considered which, though not always theoretically optimal, often yields much better algorithms in practice.

In this chapter we are mainly interested in decidability results, and thus we do not present such algorithms. However, most of the positive results presented here have been turned into efficient procedures.

Implementability of timed systems

Implementing mathematical models on physical machines is an important step for applying theoretical results to practical examples. This step is well-understood for many untimed models that have been studied (e.g. finite automata, pushdown automata). In the timed setting, while timed automata are widely-accepted as a framework for modeling real-time aspects of systems, it is known that they cannot be faithfully implemented on finite-speed central processing units (CPUs) [CAS 02]. Studying the "implementability" of timed automata is thus a challenging problem which has obvious theoretical and practical interest.

Structure of the chapter

Section 9.2 is elaborated around the standard region graph construction for timed automata. After describing it precisely, we will present other models with a finite-state abstraction of their state space, and decidability results for temporal logics resulting from these abstractions.

When considering extensions of Petri nets with time, we obtain models that are infinite "in two directions". We present in section 9.3 the setting of timed Petri nets, for which an equivalence relation with an infinite number of equivalence classes can be used to prove decidability results.

Finally, we discuss in section 9.4 implementability issues in timed automata, and present a recent approach based on a parametric semantics for timed automata.

9.2. Construction of the region graph

To decide reachability properties in timed systems, many positive results are based on the construction of a finite-state automaton that abstracts the timed system. The construction ensures that checking whether a configuration (or a set of configurations) is reachable in the timed system is equivalent to checking whether a state (or a set of states) is reachable in the finite automaton. More precisely, this construction often yields a finite state automaton that precisely accepts all the untimed words that can be obtained from the timed words accepted by the timed automaton by dropping out the time components. The standard way to obtain such an abstraction is the construction of an equivalence relation of finite index (i.e. with finitely many equivalence classes) over the configurations. To be compatible with reachability checking, this equivalence must ensure that from two equivalent configurations, the same behaviors will be possible. In timed systems, there are two types of transitions, time elapsing, and discrete firing of a transition: if from a configuration, it is possible to delay (resp. to take a transition), then this can occur from an equivalent configuration, and the two configurations resulting from the two moves are then also equivalent. However, precise delays need not to be respected, and thus the equivalence will only be a *time-abstract bisimulation* (unlike timed bisimulations defined in Chapter 4).

9.2.1. *Timed automata*

We describe a notion of regions initially presented in [ALU 90, ALU 94] for the decision of the reachability of a location in a timed automaton.

The equivalence relation

For timed automata, an equivalence relation (with finite index) verifying the above properties always exists, and it can be defined as follows. Let $\mathcal{A}$ be a timed automaton with set of clocks X and maximal constant M. Without loss of generality, we assume that all constants of $\mathcal{A}$ are integers, this can be obtained by multiplying all the constants by the same value. Two configurations (q, v) and (q', v') are equivalent if $q = q'$ and $v \equiv_M v'$, where the relation $v \equiv_M v'$ holds whenever for each clock $x \in X$,

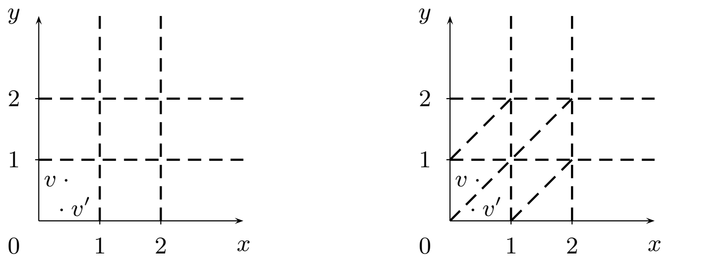
1) $v(x) > M \iff v'(x) > M$;

2) if $v(x) \leq M$, then[1] $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, and $\langle v(x) \rangle = 0 \iff \langle v'(x) \rangle = 0$, and for each pair of clocks (x, y);

3) if $v(x) \leq M$ and $v(y) \leq M$, then $\langle v(x) \rangle \leq \langle v(y) \rangle \iff \langle v'(x) \rangle \leq \langle v'(y) \rangle$.

As M is defined as the maximal constant appearing in $\mathcal{A}$, the two first conditions imply that two equivalent valuations satisfy exactly the same clock constraints of the

1. $\lfloor v(x) \rfloor$ (resp. $\langle v(x) \rangle$) denotes the integral part of $v(x)$ (resp. its fractional part).



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

Figure 9.1. *Partitions of the upper-right quarter of the plane*

timed automaton. The third 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 denote the set of regions obtained in this way as $\mathcal{R}_M(X)$. More formally, we have, given two clock valuations $v, v' \in \mathbb{R}_+^X$,

$$v \equiv_M v' \Rightarrow \left\{ \begin{array}{l} \text{for any constraint } g \text{ in } \mathcal{A},\ v \models g \iff v' \models g \\ \forall d \in \mathbb{R}_+, \exists d' \in \mathbb{R}_+ \text{ s.t. } v + d \equiv_M v' + d' \end{array} \right. \tag{9.1}$$

We can verify that the number of regions is bounded by $n! \cdot 2^n \cdot (2M+2)^n$, where n denotes the number of clocks. Indeed, a region is characterized by specifying:

(i) a mapping from clocks to the set of so-called one-dimensional regions $\{[0,0],]0,1[, [1,1], \ldots,]M-1, M[, [M,M],]M, +\infty[\}$;

(ii) for each pair of clocks x and y such that, according to point (i), both clocks have non-nul fractional parts and are bounded by M, whether $\langle x \rangle$ is less than, equal to, or greater than $\langle y \rangle$.

Point (i) yields $(2M+2)^n$ choices as there are $2M+2$ one-dimensional regions. Point (ii) is characterized by a subset of representative clocks (thus at most 2^n choices) and an order on clocks (at most $n!$ choices). Indeed, we can obtain the strict/non-strict order on fractional values clocks from the order on clocks by collapsing it on the representative clocks.

To illustrate this definition, we consider the case of two clocks x and y, with maximal constant 2. A valuation can then be understood as a point in the quarter of the plan depicted in Figure 9.1(a). The partition depicted in Figure 9.1(a) respects all constraints defined with integral constants smaller than or equal to 2, but the two valuations v and v' are not equivalent due to time elapsing (item 3 above): indeed, if we

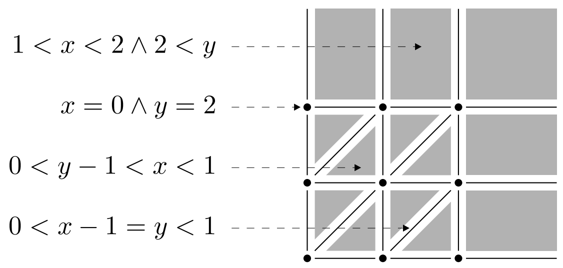


Figure 9.2. *The 44 regions in two dimensions with maximal constant* 2.

let some time elapse from the valuation v, we will first satisfy the constraint $y = 1$ and then $x = 1$, while it will be the converse from the valuation v'. Hence, the possible behaviors from v and v' are different. More precisely, property (9.1) is not satisfied: there exists a delay d such that $v + d$ satisfies the constraint $x < 1 \wedge y = 1$, while there exists no such delay for valuation v'. To handle time elapsing constraints induced by time-elapsing, condition 3) refines the partition of Figure 9.1(a) by adding diagonal lines. The resulting partition is given on Figure 9.1(b) and is a time-abstract bisimulation.

Following the above characterization of regions, they can be represented by the linear equalities and inequalities satisfied by the valuations they represent. All the regions of our example are represented on Figure 9.2. Note that there are different types of regions: regions reduced to a single element (corners), regions composed of a segment, regions composed of a half-line, regions composed of a triangle, etc. Examples of constraints associated with regions are given on this figure.

Construction of the region automaton

We present now how regions allow a finite-state automaton to be built that accepts the untimed language of the timed automaton. States of this automaton are pairs (q, R) composed of a control state of $\mathcal{A}$ and of a region of $\mathcal{R}_M(X)$. There exists a transition $(q, R) \xrightarrow{a} (q', R')$ if, and only if, there exists a transition $q \xrightarrow{g,a,r} q'$ in $\mathcal{A}$, a valuation v in the equivalence class R, and a non-negative duration $t \in \mathbb{R}_+$ such that $v + t \models g$ and $v' = (v + t)[r \leftarrow 0]$ belongs to the equivalence class R'. Note that in this definition, a transition in the region automaton represents an action transition preceded by a delay transition. We could also define transitions in the region automaton to explicitely represent delay transitions in the timed automaton in order to express qualitative properties on time elpasing. We denote by $\mathcal{R}(\mathcal{A})$ the resulting finite-state automaton, which we call *the region automaton*. Recall that operator $Untime$ projects a timed language on its untimed component. We can easily prove the following property:

PROPOSITION 9.1 *Let $\mathcal{A}$ be a timed automaton. We have:*

$$\mathcal{L}(\mathcal{R}(\mathcal{A})) = Untime(\mathcal{L}(\mathcal{A}))$$

As a consequence, we obtain:

THEOREM 9.2 ([ALU 94]) *The two following problems are PSPACE-complete:*

– Checking the emptiness of the language of a timed automaton;

– Checking the reachability of a location in a timed automaton.

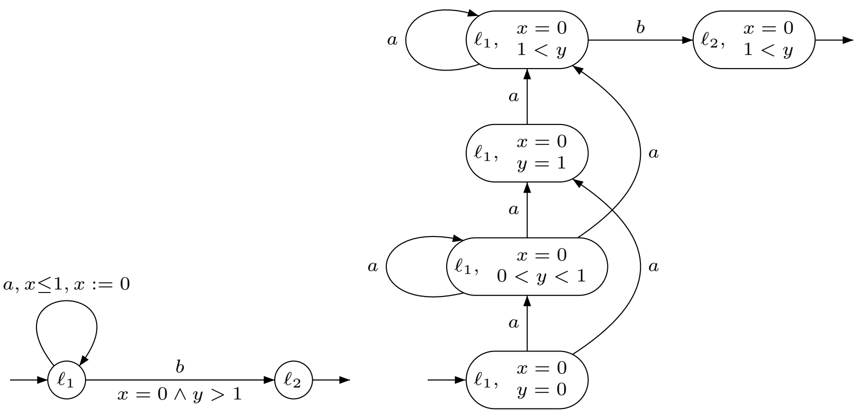
These problems are already PSPACE-hard for a fixed number k of clocks, with $k \geq 3$.

These two problems can be reduced to each other by considering accepting locations. PSPACE-membership can be deduced from the construction of the region automaton of a timed automaton, considering the bound given above on the number of regions. Indeed, the number of locations of the region automaton is exponential in the size of the timed automaton, and can thus be stored in polynomial space. Then, an NPSPACE procedure simply guesses the path allowing the location to be reached, and checks that the path is correct with polynomial space. As there are exponentially many locations in the region automaton, the length of this path is at most exponential, and can thus also be stored in polynomial space. We conclude with Savitch's theorem. To prove the PSPACE-hardness, one can encode the behavior of a linearly space bounded Turing machine on a given input. In this encoding, clock values are used to represent the content of the tape during the execution of the Turing machine.

To conclude this section, we illustrate the construction of the region automaton with an example. Consider the timed automaton depicted on Figure 9.3, which has two clocks x and y. The region automaton resulting from the previous definitions is depicted on Figure 9.3(b). In each of its location, the location of the timed automaton and the constraint corresponding to the region are given. The language of this automaton is a^*a^2b. In particular, this shows that in order to reach the accepting location in the timed automaton, one the loop should fired twice around location ℓ_1.

9.2.2. *Other timed models with finite-state abstractions*

Region construction is a very standard tool for proving decidability results in timed systems. In the chapter, we sketch other region constructions for extensions of timed automata or for other timed models.



$a, x\leq 1, x := 0$
ℓ_1
b
$x = 0 \wedge y > 1$
ℓ_2
ℓ_1, $x = 0$, $1 < y$
b
ℓ_2, $x = 0$, $1 < y$
ℓ_1, $x = 0$, $y = 1$
ℓ_1, $x = 0$, $0 < y < 1$
ℓ_1, $x = 0$, $y = 0$
a

(a) A timed automaton

(b) Its region automaton

Figure 9.3. *A timed automaton and its region automaton*

Extensions of timed automata

Timed automata constitute an extension of finite-state automata equipped with clocks, in which transitions are labeled by conditions on clock values, and by operations of updates on clock values. In the original model of Alur and Dill, conditions are Boolean combinations of comparisons of a clock with a constant and updates are simply resets to zero.

These two aspects can be extended in different ways, and several works have studied such extensions. We briefly present here some of the results established in [BOU 04].

In this work, conditions are extended with so-called *diagonal constraints*. Such constraints are of the form $x - y \sim c$, where x and y are clock variables and $\sim \in \{\leq, <, =, >, \geq\}$. They are called diagonal as they involve the difference between two clocks.

Regarding updates, they consider several extensions of the standard zero reset:

– reset to a constant value different from zero: $x := c$;

– reset to the value of another clock: $x := y$;

– incrementation: $x := x + 1$;

– decrementation: $x := x - 1$;

$\mathcal{U}_0(X)\cup$	Diagonal free constraints	General constraints
$x := c, x := y$	PSPACE-complete	PSPACE-complete
$x := x + 1$		Undecidable
$x := x - 1$	Undecidable	

Table 9.1. *Decidability status of extensions of timed automata*

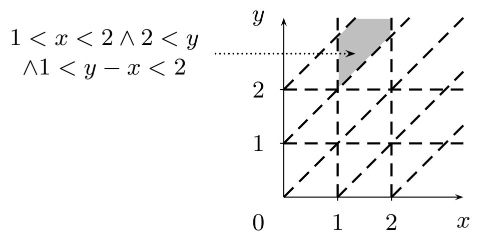


Figure 9.4. *Partition compatible with the presence of diagonal constraints*

Denoting $\mathcal{U}_0(X)$ as the standard set of updates allowed in timed automata, some of the results of [BOU 04] are summarized in Table 9.1. Among them, it is interesting to note that while adding decrementation to updates always leads to undecidability, it is possible to allow incrementation if diagonal constraints are not allowed.

To obtain the decidability results presented in this table, the authors exhibit different conditions of compatibility of a set of regions with a set of updates and/or a set of clock conditions. In some cases, it is possible to explicitly build regions verifying these compatibility conditions. We describe such a case below. However, in more involved cases, the author expresses the existence of compatible regions as the solution of a system of diphantine equations, and prove that this sytem always admits a solution. Note that to prove some of these results, we could also reduce the problem to standard timed automata by using automata translations. For instance, this can be done for diagonal constraints [BÉR 98].

Regions in presence of diagonal constraints. We describe here a construction of regions for a system composed of two clocks x and y, containing diagonal constraints, and whose maximal constant is 2. A possible partition of the set of clock valuations is described on Figure 9.4. We can easily verify that this partition ensures property (9.1), and that it is compatible with the reset operation (the result of a region by a reset is still a region). In the general case of a set of clocks X and a maximal constant M, in addition to the constaints defined in subsection 9.2.1, we have to consider constraints of the form $x - y \leq c$, for any pair of clocks x, y and any $c \in \{0, 1, \ldots, M\}$.

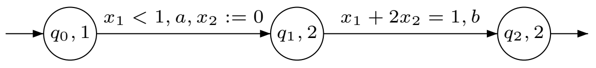


Figure 9.5. *An interrupt timed automaton*

Interrupt timed automata

The model of *interrupt timed automata* has recently been introduced in [BÉR 09] to model multi-task systems with interruptions.

We recalled in Chapter 4 that the class of hybrid automata is undecidable. An interesting subclass is the class of stopwatch automata, which extends timed automata by allowing clock variables to be frozen. However, the reachability problem is also undecidable for this class.

Interrupt timed automata form a subclass of stopwatch automata, where the real valued variables (with rate 0 or 1) are organized along priority levels. As proved in [BÉR 09], untiming languages accepted by these models yields regular languages with the effective construction of a region automaton.

We will not precisely describe the model, but will indicate on how it operates and comment on the proof of decidability. In an interrupt timed automaton, the set of locations is partitioned into interrupt levels, denoted by integers $1, \dots, n$. In addition, a unique clock is associated with each level, denoted by x_i for level i, and in a location of level i, only clock x_i is active (*i.e.* evolves when time elapses). The clocks from lower levels are suspended, as stopwatches, and those from higher levels are not relevant. Regarding transitions, the guard of a transition leaving a location of level k can be any linear constraint involving clocks $x_1, \dots, x_k$. Finally, concerning updates, if the transition enters a location of level k' with $k < k'$, then the new clocks $x_{k+1}, \dots, x_{k'}$ are simply reset to zero. Other clocks can reset using linear constraints involving clocks from *strictly lower* levels. If the transition does not strictly increase the level, then only clocks $x_1, \dots, x_k$ can be reset, using linear constraints involving clocks from strictly lower levels.

As an example, an interrupt timed automaton considered in [BÉR 09] is depicted in Figure 9.5. In this figure, the level of locations is indicated beside its name. Location q_0 is the initial location with level 1. Locations q_1 and q_2 are on level 2, and location q_2 is the final location. As there are two levels, there are thus two interrupt clocks x_1 and x_2. Consider an execution in this model. The constraint $x_1 < 1$ on the transition labeled by a implies that it is fired after a delay $1 - \tau$, with $0 < \tau \leq 1$, reaching a configuration (q_1, v) where v is the clock valuation defined by $v(x_1) = 1 - \tau$ and $v(x_2) = 0$. Then, the second transition is fired after a delay τ' which must satisfy $(1 - \tau) + 2\tau' = 1$. This yields $\tau' = \frac{\tau}{2}$ and thus the timed language accepted by this interrupt timed automaton is $L = \{(a, 1-\tau)(b, 1-\frac{\tau}{2}) \mid 0 < \tau \leq 1\}$. Note in particular that this language *cannot* be accepted by a timed automaton.

The construction of the region automaton for interrupt timed automata extends the construction of [ALU 94] as follows. For timed automata, we defined regions as partitions of the set of clock valuations that respect constraints appearing in the timed automaton, and that are compatible with time elapsing and reset. In the setting of interrupt timed automata, clock constraints are more general as they involve linear constraints on interrupt clocks. Similarly, updates are also more general. Thus, we define here the set E_k of linear constraints for each level k, which can be built in the interrupt timed automaton on level k (this includes guards of level k, but also combinations with updates of higher levels). Then, regions associated with location q are characterized by a total preorder on E_k, for each $1 \leq k \leq \lambda(q)$, where $\lambda(q)$ denotes the level of q. Intuitively, this is used to ensure that two clock valuations belonging to the same region give the same order to all the linear constraints in which they could be evaluated. This implies that the same time elapsing and the same discrete transitions are possible. This can be used to prove the correctness of the construction of a region automaton for interrupt timed automata. As for timed automata, this automaton is time-abstract bisimilar to the original model, and thus it recognizes the untiming of the language of the interrupt timed automaton. The complete description of this procedure, and its application on the example of Figure 9.5 can be found in [BÉR 09].

Time Petri nets

We conclude this subsection with the setting of *time Petri nets*, a model presented in Chapter 4. Note that the set of states of a time Petri net may be infinite for two reasons: on one hand because it can admit an unbounded number of (untimed) markings and, on the other hand, because its semantics is a timed transition system which involves unbounded (and dense) clock valuations.

Most verification problems are undecidable for time Petri nets, and thus for verification issues of this model, one considers *bounded* time Petri nets, i.e. time Petri nets that admit a finite number of (untimed) reachable markings. We will present in the next section decidability results for timed Petri nets, which, however, also have an infinite state space "for two reasons".

Decidability results for bounded time Petri nets rely on the construction of the so-called *state class graph* [BER 83, BER 91], which can be seen as the construction of a region graph. This construction preserves the reachable markings and the LTL properties. Other graph constructions have been proposed that decide the reachability of a configuration of the time Petri net, or that decide whether a CTL property is satisfied [BER 03].

Let $\mathcal{N} = (P, T, \Sigma_\varepsilon, Pre, Post, M_0, \lambda, I)$ be a time Petri net. A *state class* is a pair (M, D) composed of a marking M on places P, and a firing domain D. The firing domain should give, for each enabled transition t, the set of delays after which the timing constraint of t is satisfied. Formally, it is given by a system of linear inequalities involving one variable per enabled transition. Two state classes (M, D) and (M', D') are said *equivalent*, denoted $(M, D) \cong (M', D')$, whenever $M = M'$ and D and D' have equal solution sets.

Notations. Let $t \in T$ be a transition. By definition $I(t)$ gives the interval of firing of t. We denote by $eft(t)$, which stands for the earliest firing time, (resp. $lft(t)$, which stands for the latest firing time) the left-bound (resp. the right-bound) of $I(t)$. In the sequel, we denote by x_t a variable associated with transition t.

The construction of the state class graph proceeds as follows:
Initialization: the initial state class is (M_0, D_0) where D_0 is defined by the following set of inequations: $\{eft(t) \leq x_t \leq lft(t) \mid {}^\bullet t \leq M_0\}$;
New Successor: let (M, D) be a state class built by the procedure. Transition t is firable from (M, D) if and only if the two following conditions are satisfied:

(i) ${}^\bullet t \leq M$ (discrete enabledness), and

(ii) the system $D \wedge \{x_t \leq x_{t'} \mid t' \neq t \wedge {}^\bullet t' \leq M\}$ admits a solution.

Let t be a transition satisfying these conditions, then we build a new state class, denoted (M', D'), which is the successor of (M, D) by the transition t. This state class is defined by $M' = M - {}^\bullet t + t^\bullet$, and D' is obtained by:

(a) $D := D \wedge \{x_t \leq x_{t'} \mid t' \neq t \wedge {}^\bullet t' \leq M\}$;

(b) for each transition u enabled at M', a new variable x'_u is introduced, obeying:
$x'_u = x_u - x_t$ if the predicate[2] $\uparrow enabled(u, M, t)$ evaluates to false,
$eft(u) \leq x'_u \leq lft(u)$ otherwise;

(c) variables x_v for $v \in T$ are eliminated by considering an existential quantification for these variables (this is possible in linear arithmetic over real numbers).

In this construction, when a new state class is built, if an equivalent state class (with respect to relation $\cong$ defined above) exists in the graph, then the two nodes of the graph are merged. The equivalence $\cong$ can be checked by transforming the systems of linear inequalities into canonical forms. Indeed, these systems are conjunctions of inequalities on differences of variables. Thus, a canonical form can be obtained by the computation of the strongest constraints, what can be obtained by a "shortest paths" algorithm, for instance the Floyd-Warshall algorithm. Note that while the time complexity of the general Floyd-Warshall algorithm is $O(n^3)$ where n denotes the number of variables, the computation of a canonical form after the above operations can be performed in time $O(n^2)$ if the previous constraints were in canonical form.

2. See Chapter 4 for the definition of this predicate.

Then, we can state the following theorem:

THEOREM 9.3 ([BER 83]) *Let $\mathcal{N}$ be a bounded time Petri net, then the state class graph of $\mathcal{N}$ is finite and recognizes the language $Untime(\mathcal{L}(\mathcal{N}))$.*

To conclude this section, we illustrate this construction with an example. We consider the time Petri nets depicted in Figure 4.12 of Chapter 4. The state class graph obtained by the procedure described above applied on this time Petri net is depicted in Figure 9.6. In this figure, we represent the marking and the constraints defining the firing domain in each node state class. On each transition, we write both the name of the transition from the original time Petri net and its label. Note that variable x_i refers to transition t_i. When no transition is enabled in a marking, we denote by $\perp$ the firing domain. Consider the transition labeled by a between markings $p_1 + p_2 + p_3$ and $p_2 + 2p_3$. This corresponds to the firing of transition t_1 of the net. The firing of this transition from this marking does not newly enable transition t_3, but it newly enables transition t_2. As a consequence, the constraint associated with t_3 has been modified, since it is deduced from the fact that at least one time-unit has elapsed to allow the firing of t_1. On the contrary, the constraint associated with t_2 is fresh, as t_2 is newly enabled. Note also that the discrete structure of the graph is different from that of the reachability of the underlying Petri net. Indeed, marking p_3 is represented twice with different constraints.

9.2.3. *Application to the decision of temporal logics*

The region construction of timed automata also has consequences on the decidability results of model checking. We present here two model-checking algorithms for timed automata based on the region automaton construction.

Model checking LTL

The model checking of LTL properties for finite-state systems was presented in subsection 7.4.2. It relies on the construction of a Büchi automaton for the LTL formula, and of its synchronized product with the automaton of the model. Following Proposition 9.1, which states that the untimed language is preserved by the region automaton construction, the same process can be applied on the region automaton, yielding the decidability of the LTL model checking for timed automata.

Model checking TCTL

The timed logic TCTL has been presented in Chapter 4. It constitutes a timed extension of the branching logic CTL. We present here an adaptation of the region automaton construction which allows to decide whether a timed automaton satisfies a property expressed as a TCTL formula.

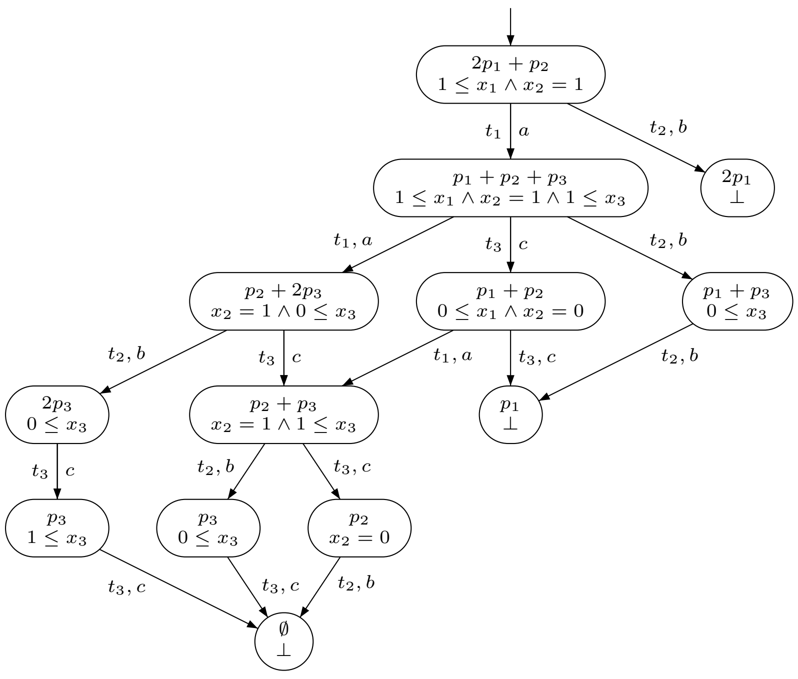


Figure 9.6. *A state class graph of a time Petri net*

THEOREM 9.4 ([ALU 93]) *Model checking* TCTL *over timed automata is decidable and the problem is PSPACE-complete.*

The model checking of TCTL formulas on timed automata can be reduced to the model checking of a CTL formula on a modified region automaton. We describe here how this procedure works. The central property of the equivalence used above to define the regions of a timed automaton is its consistency with TCTL formula: let $\mathcal{A} = (\Sigma, X, Q, q_0, \Delta)$ be a timed automaton, $\equiv_M$ the associated equivalence relation, $q \in Q$ a state and $v \equiv_M v'$ two equivalent valuations. Then for each formula ϕ of TCTL, $(q, v) \models \phi \iff (q, v') \models \phi$. The proof of this property relies on a notion of equivalence between runs, extending the equivalence between valuations. Then, for model checking, a construction similar to that for the region automaton is performed, which additionally takes into account the formula ϕ that $\mathcal{A}$ must satisfy:

– a new clock z_ϕ (not in X) is added to measure delays associated with subformulas of ϕ. We set $X^* = X \cup \{z_\phi\}$;

– recall that M is the maximal constant associated with $\mathcal{A}$. In order to take into account the constants appearing in the time constraints of the subformulas, we denote the largest of these constants as M_ϕ and we set $M' = \max(M, M_\phi)$.

Then a region automaton $H_\mathcal{A}$ is built from $\mathcal{A}$ with X^* as set of clocks and M' as maximal constant. States are thus elements of the form (q, R), where q is a location of $\mathcal{A}$ and $R \in \mathcal{R}_{M'}(X^*)$ is a region involving one more component than regions on X, corresponding to the values of the special clock z_ϕ. In this construction, the case of boundary regions must also be handled by modifying some transitions (we omit these technical details here). The last step before applying the CTL labeling algorithm is to add atomic propositions $p \sim c$ to label the states of $H_\mathcal{A}$: a state (q, R) is labeled by $p \sim c$ if the value of z_ϕ in the region R satisfies the constraint $z_\phi \sim c$. For instance, to handle a formula like $\phi : AF_{<3}P$, meaning that atomic proposition P will hold on all runs before three time units, we transform the condition for a configuration $(q, v) \models \phi$ into $(q, [v, 0]) \models AF(P \wedge p < 3)$ in $H_\mathcal{A}$.

In other words, from a state where the value of z_ϕ is equal to 0, a state can always be reached where P holds and at the same time the value of z_ϕ is less than 3. The construction is illustrated on the timed automaton of Figure 9.3. This timed automaton has two clocks x and y and 1 as maximal constant 1. We also choose 1 for M_ϕ, so we can deal with formulas using 0 or 1 as constants. The region automaton $H_\mathcal{A}$ is thus built with $M = 1$. For the sake of simplicity, consider only formulas with one level (no nested formulas). Then the clock z is never reset (as clock y in this example) and thus the region automaton $H_\mathcal{A}$ is obtained from the region automaton depicted in Figure 9.3(b) by letting z be equal to y in all regions.

To conclude this section, note that this technique can be adapted to obtain a procedure for the model checking of the timed logic L_ν [LAR 95].

9.3. Handling infinite abstractions

The different settings presented in the previous section share the property of admitting an equivalence relation of a finite index. This yields a finite number of regions and thus allows decidability results to be deduced. We present here a setting in which the number of regions is infinite, but still yields a decidability result, through additional techniques of infinite state systems (here, well quasi orders). This setting concerns the model of timed Petri nets, an extension of Petri nets which model timed systems. We do not recall here the model as it was presented in Chapter 4.

A lazy semantics

As for the model of time Petri nets, the set of reachable states of a time Petri net may be infinite for two reasons: on one hand because it can admit an unbounded number of (untimed) markings and, on the other hand, because its semantics involves unbounded (dense) clock valuations. However, as we will see, several problems are decidable for this model. This positive property partly follows from the fact that the semantics of time Petri nets is urgent, while the semantics of timed Petri nets is lazy. In time Petri nets, when a transition is enabled, it cannot get disabled by time elapsing. On the opposite, in timed Petri nets, one can miss the firing of a transition by time elapsing. This induces a monotony property for timed Petri nets, not satisfied by time Petri nets, that is central in the decidability results we present in this section.

9.3.1. *A symbolic representation for timed Petri nets*

Let $\mathcal{N} = (P, T, \Sigma_\varepsilon, Pre, Post, M_0, \Lambda)$ be a timed Petri net where the bounds of intervals are in $\mathbb{N}_{\geq 0} \cup \{\infty\}$. There is no loss of generality in assuming that finite bounds of the net are integers (otherwise we refine the granularity of the regions).

Definition of the Coverability problem. Let N be a finite set of markings with integral ages (again, we could pick rational numbers and refine the granularity). By $N^\uparrow$, we denote the upward closure of N, *i.e.*, the set $\{\nu \mid \exists \nu' \in N,\ \nu' \leq \nu\}$ (where the order is the standard order of $Bag(P \times \mathbb{R}_+)$). The *coverability problem* for $\mathcal{N}$ and set of configurations N asks whether there exists a path in $\mathcal{N}$ from ν_0, the initial configuration of $\mathcal{N}$, to some $\nu \in N^\uparrow$. We prove the following result.

THEOREM 9.5 (Coverability Problem [ABD 01]) *The coverability problem is decidable for timed Petri nets.*

In order to prove this theorem, we introduce the notion of region for timed Petri nets. Such a construction has been done for example in [MAH 05] for timed Petri nets, and has been used recently in several other contexts [OUA 04, OUA 05, LAS 05], and extended to timed Petri nets with read-arcs in [BOU 08a]. By $\max$ we denote the maximal integer appearing in the bounds of intervals of the net and in the ages of the tokens in the configurations of N.

DEFINITION 9.6 *A region R for $\mathcal{N}$ is a sequence $a_0 a_1 \ldots a_n a_\infty$ where $n \in \mathbb{N}_{\geq 0}$, for all $0 \leq i \leq n$, $a_i \in Bag(P \times \{0, 1, \ldots, \max\})$ with $size(a_i) \neq 0$ if $i \neq 0$, and $a_\infty \in Bag(P \times \{\infty\})$.*

We first informally explain the semantics of a region. Given the bag of tokens defining a configuration, we obtain its associated region as follows. We put in a_∞ all the tokens whose ages are strictly greater than $\max$ and forget their ages. We then put in a_0 the tokens with integral ages and add the information about their ages. Finally, we order the remaining tokens depending on the fractional part of their ages in $a_1, \ldots, a_n$, forget their fractional part, and only store the integral part of their ages. Hence n is the number of different positive fractional values for ages of the remaining tokens. For instance, consider the bag of tokens $(p, 1) + (p, 2.8) + (q, 0.8) + (q, 5.1) + (r, 1.5)$. Then, if the maximal constant is 4, its region encoding will be $a_0 a_1 a_2 a_\infty$ where $a_0 = (p, 1)$ (because there is a single token with integral age), $a_\infty = (q, \infty)$ (because the age of token $(q, 5.1)$ is 5.1, hence above the maximal constant), $a_1 = (r, 1)$ (among all fractional parts, 0.5 is the smallest one), and $a_2 = (p, 2) + (q, 0)$ (all tokens with fractional part 0.8).

We now define more formally the semantics of the regions. Let ϕ be the mapping from $\mathbb{R}_+$ to $\{0, 1, \ldots, \max, \infty\}$ defined by: if $x > \max$ then $\phi(x) = \infty$ else $\phi(x) = \lfloor x \rfloor$. We extend ϕ to $P \times \mathbb{R}_+$ by $\phi((p, x)) = (p, \phi(x))$ and to $Bag(P \times \mathbb{R}_+)$ by linearity. Let us recall the order defined on finite multisets (also called bags). An element $m \in Bag(Z)$ is a mapping with finite domain from Z to $\mathbb{N}$. Given $m, m' \in Bag(Z)$, $m \leq m'$ holds if, and only if, for all $z \in Z$, we have $m(z) \leq m'(z)$. To ease the reading, given $z \in Z$ and $m \in Bag(Z)$, we may also write $z \leq m$ whenever $m(z) \geq 1$.

Let $R = a_0 a_1 \ldots a_n a_\infty$ be a region. Then $[R]$ is a set of configurations ν such that there exist $\nu_1, \nu_2, \ldots, \nu_n, \nu_\infty$ belonging to $Bag(P \times \mathbb{R}_+)$ with:

– $\nu = a_0 + \nu_1 + \nu_2 + \ldots + \nu_n + \nu_\infty$;

– $\forall 1 \leq i \leq n, \phi(\nu_i) = a_i$, and $\phi(\nu_\infty) = a_\infty$;

– $\forall 1 \leq i \leq n, \forall (p, x) + (q, y) \leq \nu_i, 0 < x - \lfloor x \rfloor = y - \lfloor y \rfloor$;

– $\forall 1 \leq i < j \leq n, \forall (p, x) \leq \nu_i, (q, y) \leq \nu_j, x - \lfloor x \rfloor < y - \lfloor y \rfloor$.

Note that every configuration ν belongs to a single region, that we write $R(\nu)$. Conversely, according to the hypothesis, elements of N have integral ages, for any $\nu \in N$, we have $[R(\nu)] = \{\nu\}$. The original coverability problem thus reduces to the coverability problem for finitely many regions, which itself reduces to solve the coverability problem for a single region R.

9.3.2. *Coverability of timed Petri nets*

We first notice that, given two regions $R = a_0 a_1 \ldots a_n a_\infty$ and $R' = a'_0 a'_1 \ldots a'_{n'} a'_\infty$, we can check whether $[R]^\uparrow \subseteq [R']^\uparrow$: the necessary and sufficient conditions are $a_0 \geq a'_0$, $a_\infty \geq a'_\infty$ and the existence of a strictly increasing mapping ψ from $\{1, \ldots, n'\}$ into $\{1, \ldots, n\}$ such that for every $1 \leq i \leq n'$, $a_{\psi(i)} \geq a'_i$.

We define a partial order between regions by $R \leq R'$ iff $[R']^\uparrow \subseteq [R]^\uparrow$. Then, using Higman's lemma [HIG 52], we can show that this is a well quasi-order, i.e., for every infinite sequence of regions $\{R_i\}_{i\in\mathbb{N}}$ there exist $i < j$ such that $R_i \leq R_j$. Indeed, each region R is a finite sequence of bags over a finite set, hence applying [ABD 01, theorem 1], the above mentioned partial order is a well quasi-order.

The algorithm for solving the coverability problem for the upward closure of a single region R then consists of iteratively computing the predecessors (by time elapsing and by discrete steps) of $[R]^{\uparrow}$. As we will see, each such predecessor is a finite union of upward closures of regions. We stop exploring the predecessors of an upward closure of a region when it is larger (for partial order $\leq$) than an already computed region. Note that all configurations reachable from $[R_2]^{\uparrow}$ are also reachable from $[R_1]^{\uparrow}$ whenever $R_1 \leq R_2$. The computation can then be seen as a finitely branching tree. To prove that it terminates, it is sufficient to prove that this tree is finite. Suppose it is not. By applying König lemma, this tree has an infinite branch. However, as $\leq$ is a well quasi-order, we will eventually obtain a region that is larger than a previous one. This leads to a contradiction. Hence, the computation tree is finite, and the computation terminates. The set of configurations N is covered by the timed Petri net $\mathcal{N}$ if and only if its initial configuration ν_0 occurs in the upward closure of some region of the tree.

Finally we explain how we compute the time and discrete predecessors of the upward closure of a region $R = a_0 a_1 \dots a_n a_\infty$.

Time predecessors

If a_0 contains a token $(p, 0)$, there is no strict time predecessor of $[R]^{\uparrow}$. Otherwise if $size(a_0) \neq 0$, then the time predecessor is $[R']^{\uparrow}$ with $R' = a'_0 a_1 \dots a_n a'_{n+1} a_\infty$ where a'_0 is the empty bag and a'_{n+1} is obtained from a_0 by decrementing by 1 the (integral) age of each token. Informally, this operation represents a (reverse) small time elapse such that no token of a_1 reaches an integral value and no token of a_∞ reaches back $\max$.

Otherwise (*i.e.*, $size(a_0) = 0$) we need to choose if tokens of a_1 will first reach an integral value or some tokens of a_∞ will first reach $\max$. It could be the tokens of a_1, a bag of tokens $b_\infty \leq a_\infty$, or both. We only illustrate this last case (which assumes $n \geq 1$). The above mentioned time predecessor is $[R']^{\uparrow}$ where $R' = a'_0 a'_1 \dots a'_{n-1} a'_\infty$ is obtained as follows.

– $a'_\infty = a_\infty - b_\infty$;

– $a'_0 = a_1 + c_\infty$ where c_∞ is obtained from b_∞ by setting the age of each token to $\max$;

– $\forall 1 \leq i \leq n-1, a'_i = a_{i+1}$.

Discrete predecessors

We pick a transition t. Recall that in the syntax of a timed Petri net, there exist two mappings Pre and $Post$ from $T \times P$ to the set $\mathcal{I}$ of closed intervals with a lower bound in $\mathbb{N}$ and an upper bound in $\mathbb{N} \cup \{\infty\}$ (we assume that the constants are integers instead of rational numbers, this can be achieved by multiplying all the constants of the system). In addition, the fact that place p is not an input place of transition t corresponds to the case where $Pre(t, p)$ is the empty interval. We introduce an alternative notation for pre- and postconditions. We define $Pre(t) = \sum_{p \mid Post(t,p) \neq \emptyset} (p, Post(t, p))$, and similarly for $Post(t)$. For instance, a transition t consuming a token in place p_1 with interval $[0, 1]$, no token in p_2 and p_3, and a token in place p_4 with interval $[2, 3]$ verifies $Pre(t) = (p_1, [0, 1]) + (p_4, [2, 3])$.

Note that given an interval I of the net and a token (p, x) belonging to some a_i for $i \in \{0, 1, \ldots, n, \infty\}$, we can compute whether, given a configuration belonging to that region, the corresponding token belongs to I. According to the property of the regions, this is independent of the choice of the configuration. We then write $(i, x) \vDash I$.

We consider the upward closure of the region $a_0 a_1 \ldots a_n a_\infty$, and want to compute its preimage by transition t. Transition t produces a bag of tokens as defined by $Post(t)$. These tokens may appear in one of the a_i's, but this is not required, they may only be in the upward closure. This constitutes an important difference with standard reachability analysis, and comes from the fact that we are interested in a coverability problem. Hence, we choose bags of tokens $\mathsf{post}_i \in Bag(P \times \{0, 1, \ldots, \max\} \times \mathcal{I})$ for every $i \in \{0, 1, \ldots, n\}$ and $\mathsf{post}_\infty \in Bag(P \times \{\infty\} \times \mathcal{I})$ such that[3]

– for all $(p, x, I) \leq \mathsf{post}_i$, $(i, x) \vDash I$;

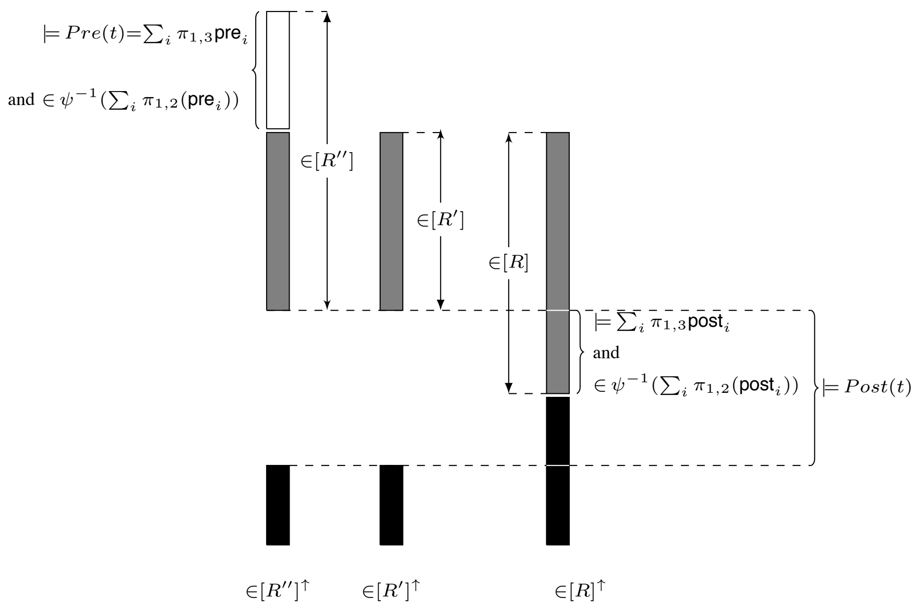
– for all $i \in \{0, 1, \ldots, n, \infty\}$, $\pi_{1,2}(\mathsf{post}_i) \leq a_i$;

– $\sum_i \pi_{1,3}(\mathsf{post}_i) \leq Post(t)$.

The bag post_i represents the tokens produced by t which "belong" to a_i. However, there may be additional tokens produced that do not appear in one of the a_i's (this is possible as we consider the upward closure of the region), which is why the two last conditions are inequalities and not equalities. Figure 9.7 illustrates the decomposition. The first condition states that the values of x, i and I in a bag (p, x, I) of post_i are coherent. The second condition states that post_i constitutes a subset of a_i. The third condition states that post_i constitutes a subset of $Post(t)$ (in the second and third conditions, projections are used as a_i contains integral values while $Post(t)$ contains intervals).

Applying this first decomposition, we build an intermediate region $R' = a'_0 a'_1 \ldots a'_{n'} a'_\infty$ by substracting $\pi_{1,2}(\mathsf{post}_i)$ from a_i for every i and deleting the item in the resulting sequence if its size is null (for $1 \leq i \leq n$).

3. $\pi_{1,2}$ (resp. $\pi_{1,3}$) projects bags onto the two first components (resp. onto the first and the third).



$\models Pre(t)=\sum_i \pi_{1,3}\mathsf{pre}_i$
and $\in \psi^{-1}(\sum_i \pi_{1,2}(\mathsf{pre}_i))$
$\in[R'']$
$\in[R']$
$\in[R]$
$\models \sum_i \pi_{1,3}\mathsf{post}_i$
and
$\in \psi^{-1}(\sum_i \pi_{1,2}(\mathsf{post}_i))$
$\models Post(t)$
$\in[R'']^\uparrow$
$\in[R']^\uparrow$
$\in[R]^\uparrow$

Figure 9.7. *Decomposition of the set of tokens for the discrete predecessor computation*

Then, to really simulate the discrete transition t, we need to initially have all tokens that are consumed by the pre-arcs. We set bags of tokens $\mathsf{pre}_i \in Bag(P \times \{0,1,\dots,\max\} \times \mathcal{I})$ for every $i \in \{0,1,\dots,n''\}$ for some integer n'', $\mathsf{pre}_\infty \in Bag(P \times \{\infty\} \times \mathcal{I})$ and a strictly increasing mapping ψ from $\{1,\dots,n'\}$ into $\{1,\dots,n''\}$ such that:

– for all $(p,x,I) \leq \mathsf{pre}_i$, $(i,x) \models I$;

– $a''_0 = a'_0 + \pi_{1,2}(\mathsf{pre}_0)$;

– $a''_\infty = a'_\infty + \pi_{1,2}(\mathsf{pre}_\infty)$,

– for every $i \in \{1,\dots,n''\}$, if there exists j such that $\psi(j) = i$ then $a''_i = a'_j + \pi_{1,2}(\mathsf{pre}_i)$, otherwise $a''_i = \pi_{1,2}(\mathsf{pre}_i)$;

– $\sum_i \pi_{1,3}(\mathsf{pre}_i) = Pre(t)$.

The bags pre_i are the tokens required by the pre-arcs of the transition. See Figure 9.7 for an illustration of the construction.

Under those conditions, the region $R'' = a''_0 a''_1 \dots a''_{n''} a''_\infty$ is a predecessor by t of $[R]^\uparrow$. Note that the constructed region R'' depends on the various choices we have made (all bags post_i, pre, and also the indices n', n'', the mapping ψ). For each of these (finitely many) choices, it gives a region that is in the preimage of R by t (indeed, take any configuration $\nu'' \in [R'']^{\uparrow}$, then quite straightforwardly, any configuration image of ν by t is in $[R]^{\uparrow}$), and all regions in the preimage by t can, of course, be obtained in that way.

Hence, time predecessors and discrete predecessors of regions are finite unions of regions, and can be effectively computed, which concludes the proof of the theorem.

9.3.3. *Survey of decidable properties for timed Petri nets*

We have presented in the previous sections how a region construction with infinitely many regions can be used to prove the decidability of the coverability problem for timed Petri nets. It is worth noticing that this model, which combines dense-time with an unbounded control structure (represented here by the tokens), is very difficult to analyze. In this section, we will present several decidability results, which, for most of them, relies on the techniques presented previously. These results are taken from [ABD 07, BOU 08a].

Token liveness

As the semantics of timed Petri nets is lazy, we can miss the firing of a transition. As a consequence, it may happen that a token can no longer be used in the firing of a transition, in which case we say that the token is not live anymore. More formally, a marking is composed of a finite set of tokens, and each token is characterized by the place p in which it lies and by its age, given as a non-negative real number x (a token is thus represented by the pair (p, x)). Let M be a marking, and (p, x) a token in M. The token (p, x) is called *live* from marking M in a timed Petri net $\mathcal{N}$ if there exists a sequence of transitions in $\mathcal{N}$, starting from M, which eventually consumes the token. Conversely, if a token is not live, then we say that it is a *dead* token. Formally, we say that the token (p, x) can be consumed in M if there exists a transition t satisfying the following properties:

– t is enabled in M; and

– $x \in Post(t, p)$.

Given a timed Petri net $\mathcal{N}$, a marking M; and a token (p, x) in M, the problem of token liveness consists of deciding whether (p, x) is live or not. This problem is called the semantic liveness of tokens. Without loss of generality, we assume that the ages of tokens composing the marking M are all integers.

Theorem 9.7 ([ABD 07]) *The semantic liveness of tokens is decidable in timed Petri nets.*

The proof of this result is proceeded by a direct reduction to (several instances of) the coverability problem. As we have seen that this problem is decidable for timed Petri nets, the result follows. We briefly describe how the reduction works.

We define the timed Petri net $\mathcal{N}'$ as follows. The set of places is obtained by substituting of place p with a new place p', not connected to any transition. The initial marking is the marking $M_{init} = M - (p, x) + (p', x)$. Finally, the set of target markings N is defined as follows. Let t be a transition such that p is an input place of t. Then we write $Pre(t) = \sum_{i=1}^{n}(p_i, I_i) + (p, I)$. We define the set of markings N_t as:

$$N_t = \left\{ \sum_{i=1}^{n}(p_i, \tau_i) + (p', x') \mid \forall i, \tau_i \in I_i \wedge x' \in I \right\}$$

This set can be represented as a finite union of regions. We then define the set N as the union of sets N_t over the transitions t having p as input place. In this construction, we move the token from place p to a fictive place p' where it cannot be used, it will only get older as time elapses. Then, we look for markings, composed of tokens with integral ages, that allow a transition involving place p to be fired. We can easily prove that the token (p, x) is live if, and only if, the set of markings N can be covered.

Boundedness

We say that an untimed Petri net is bounded if, and only if, there exists a bound $k \in \mathbb{N}$ such that all its reachable markings have at most k tokens in each place. In particular, a bounded Petri net has a finite reachability set. For timed Petri nets, we will also be interested by bounding the number of tokens. However, this will not yield a finite reachability set because of the ages of the tokens.

Two notions of boundedness are distinguished, whether dead tokens are counted or not:

– syntactic boundedness: does a bound $k \in \mathbb{N}$ exist such that all reachable markings are composed of at most k tokens?

– semantic boundedness: does a bound $k \in \mathbb{N}$ exist such that all reachable markings are composed of at most k *alive* tokens?

The following theorem is stated in [ABD 07]:

THEOREM 9.8 ([ABD 07]) *The syntactic boundedness is decidable for timed Petri nets. The semantic boundedness is undecidable for timed Petri nets.*

The undecidability result is proven by a reduction in the problem of space boundedness for lossy counter machines. Details can be found in [MAH 05]. For the decidability result, we consider an algorithm similar to the Karp-Miller algorithm [KAR 69] to solve the boundedness problem for Petri nets. This algorithm builds a tree labeled by markings and, when a new marking is added, checks whether one of the ancestors is strictly dominated. If this is the case, then the number of reachable markings is unbounded. We can consider a similar algorithm, in which nodes of the trees are labeled by regions. The termination of the algorithm is guaranteed due to the fact that the inclusion ordering on regions is a well-quasi-ordering (see subsection 9.3.2).

Zenoness

Zeno behaviors are executions that perform an infinite number of actions in a finite amount of time. Such time-convergent behaviors are unrealistic in practice. For timed automata, the existence of such behaviors can be decided using the region graph. One can also consider this problem for timed Petri nets:

– zenoness: given a timed Petri net $\mathcal{N}$ and a marking M, does a Zeno execution in $\mathcal{N}$ exist that starts in M?

– universal zenoness: given a timed Petri net $\mathcal{N}$ and a marking M, are all infinite runs of $\mathcal{N}$ issued from M Zeno runs?

Note that the negation problem is the problem of the existence of an infinite non-Zeno execution.

The following theorem is stated in [ABD 07]:

THEOREM 9.9 ([ABD 07]) *The Zenoness is decidable for timed Petri nets. The universal Zenoness is undecidable for timed Petri nets.*

We will not give the details of the proofs of these results as they rely on sophisticated techniques of Petri nets. More precisely, the decidability result uses a characterization of the set of markings admitting an infinite computation in an extension of standard Petri nets, which is a subclass of transfer nets.

Extension to read arcs

We conclude with an extension of our first result on the coverability. The construction we have presented can be extended to the setting of timed Petri nets with read arcs. Read arcs allow a transition to test the presence of some tockens in a place, without consuming them (in particular their age is left unchanged). It can be shown that read arcs allow the expressiveness of timed automata to subsumed (see [BOU 08a]).

THEOREM 9.10 ([BOU 08a]) *The coverability problem is decidable for timed Petri nets with read arcs.*

9.4. Robustness issues in timed systems

In this section, we present issues related to the implementation of timed systems, and more specifically a recent approach proposed for the class of timed automata.

9.4.1. *Motivations*

The verification of real-time systems is now a well-established procedure, with efficient algorithms and scalable tools. However, the timed models used suffer from mathematical idealization, which makes their implementation problematic. For instance, timed automata are governed by an idealized, theoretical semantics, and their implementation on real hardware could fail to satisfy their mathematically proved properties. More precisely, timed automata assume that clocks are continuous and infinitely precise, while CPU have a finite frequency and are digital.

A well-known difficulty is the case of so-called Zeno behaviors, in which an infinite number of actions can occur in a finite amount of time, which naturally correspond to unrealistic physical behaviors. The difficulties of implementation for timed systems are, however, not limited to the non-Zenoness property. As an illustration, we recall on Figure 9.8 a timed automaton presented in [CAS 02]. In this example, location 4 is considered to be a bad location that should be avoided. Thus, the system should be able to perform infinite executions around locations 1, 2, and 3. It can be shown that infinite executions admitted by this timed automaton must go "faster and faster". More precisely, one cycle through these three locations must last exactly one time unit because of the timing constraints on x between locations 1 and 2. In addition, before firing the last transition of the cycle (from 3 to 1), the execution must let some time elapse (because of the constraint on z). We denote this value by d_i for the i-th execution of the loop. Then, valuation v_i obtained when entering location 1 after the i-th loop is defined by $v_i(x) = \sum_{j \leq i} d_j$, $v_i(y) = 0$ and $v_i(z) = d_i$. As a consequence, in order to obtain an infinite execution, delay d_i must be choosen so as to verify the inequality $\sum_{i \geq 1} d_i < 1$. For instance, a possible execution is obtained by letting $d_i = \frac{1}{2^{i+1}}$. As a consequence, the delays between the last two transitions of the cycle must converge towards 0. In particular, no discrete-time execution can verify this property, no matter how fine the granularity. As a consequence, any implementation of such an automaton will eventually fail to satisfy that property, and will finally reach location 4.

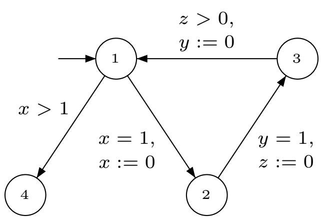


Figure 9.8. *A timed automaton admitting no discrete-time implementation*

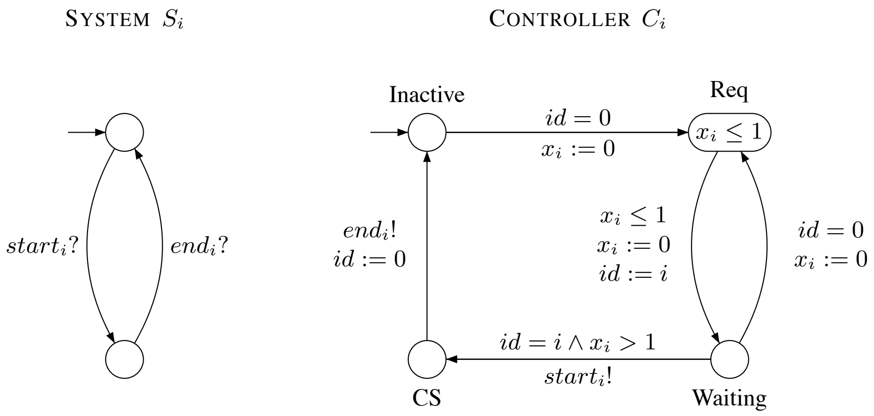


Figure 9.9. *Timed automata models for the Fischer protocol*

Consider the example of the Fischer protocol for ensuring mutual exclusion: two systems S_1 and S_2 want to access a critical section; they are supervised by two controllers in charge of ensuring that they will not simultaneously access the critical section. Figure 9.9 represents a system and its controller. We can show, using a verification tool such as Uppaal [BEH 04], HyTech [HEN 97], or Kronos [DAW 96], that mutual exclusion is ensured by these controllers. Intuitively, the protocol works as follows. A shared integer variable id is used to store the identifier of the process allowed to enter the critical section. More precisely, this variable initially has value 0. Then, when process S_i moves from location Req to location Waiting, it sets this variable to value i, and checks that $id = i$ when it enters location CS. In addition to this untimed rule, timing constraints are used to ensure that a process cannot decide to enter the critical section (by setting variable id to its identifier) when another process is already in its critical section.

However, this property greatly depends on the infinite precision (i.e. exact) of the model of timed automata. In particular, it relies on the fact that guard $x_i > 1$

is strict. Indeed, it ensures, combined with invariant $x_i \leq 1$ of location Req, that when a process S_i can reach its critical section, then no other process S_j can lie in location Req. Thus, relaxing constraing $x_i > 1$ in constraint $x_i \geq 1$ results in an incorrect system, thus proving its brittleness. For instance, if clocks are digital, or if clocks are not perfectly synchronized, or if there are some communication delays between systems, or again if delays of write/read actions slow down the system, then transitions may be performed at time stamps that are not allowed in the mathematical model. Of course, such imprecisions are unavoidable in physical devices, and thus the mutual exclusion can be violated.

9.4.2. *A parametric semantics to handle imprecisions of clock valuations*

Following the example of the Fischer protocol, some properties related to the idealization of timed models cannot be satisfied by physical devices. We list some of them:

1) perfect synchronization of the different components of the system, inputs reception without delay, output production without delay;

2) infinite precision of clocks:

3) instantaneous execution of actions (infinite speed of the system while evaluating guards, and choosing which action to fire).

A real system, when it simulates the behavior of a timed automaton, cannot fire transitions in zero time. Similarly, the variables it manipulates have a finite precision, and synchronizations between processes are not instantaneous. The standard hypothesis allowing these issues to be ignored consists of assuming that the delays related to computation times or message exchanges are negligible with respect to constants used in the guards and invariants of the timed automaton. However, this approach does not rely on any mathematical justification.

In order to handle the different imperfections of a real system, a new semantics, called the AASAP ("almost as soon as possible") semantics, was introduced in [DEW 05] for timed automata. This semantics relaxes the hypotheses that were too strong in the classical semantics. The AASAP semantics can be seen as a relaxation of the standard ASAP ("as Soon As Possible") semantics. Formally, these two semantics are defined as timed transition systems. We do not present them here, but rather describe their main characteristics. We refer the reader to [DEW 05] for a complete presentation.

The notion of timed simulation, introduced in Chapter 4 for timed transition systems, is the key tool to compare semantics. Indeed, such relations preserve safety properties, but more generally, they preserve properties expressing a universal quantification over executions, such as LTL [PNU 77] or ACTL [BRO 88].

The ASAP semantics requires a transition to be fired or to react to an input *as soon as possible*. In the AASAP semantics, a parameter Δ is given, and intuitively the system should react within Δ time units. The main characteristics of this semantics can be summarized as follows:

– an enabled transition does not need to be fired instantaneously, but must be fired within Δ time units;

– the durations of emission and reception of messages are taken into account: a distinction is made between the emission of a signal by a component and its reception by another component. The delay between these two actions is bounded by Δ;

– the precision of clocks is relaxed: guards of transitions are enlarged by Δ.

The AASAP semantics of $\mathcal{A}$ for parameter Δ is denoted $[\![\mathcal{A}]\!]^{\mathsf{AASAP}}_{\Delta}$, and verifies the following property:

PROPOSITION 9.11 ("Faster is better" [DEW 05, theorem 3]) *Let $\mathcal{A}$ be a timed automaton, and $\Delta_1, \Delta_2 \in \mathbb{Q}_{>0}$ such that $\Delta_2 \leq \Delta_1$, then the timed transition system $[\![\mathcal{A}]\!]^{\mathsf{AASAP}}_{\Delta_2}$ is simulated by $[\![\mathcal{A}]\!]^{\mathsf{AASAP}}_{\Delta_1}$, denoted by $[\![\mathcal{A}]\!]^{\mathsf{AASAP}}_{\Delta_2} \sqsubseteq [\![\mathcal{A}]\!]^{\mathsf{AASAP}}_{\Delta_1}$.*

Intuitively, this means that increasing the precision of the system reduces the set of admissible behaviors. Thus, with respect to the verification of universal path properties, if the model $[\![\mathcal{A}]\!]^{\mathsf{AASAP}}_{\Delta_1}$ verifies the property, then so does $[\![\mathcal{A}]\!]^{\mathsf{AASAP}}_{\Delta_2}$.

In order to build a link with the implementation of a timed automaton, a *program semantics* is introduced, representing the behavior of a real execution platform. We describe the model considered for this platform. The procedure simulating the timed automaton repeatedly executes the following cycle of instructions:

1) the current time is read in the clock register of the processor and stored in a variable T;

2) the list of input signals is updated: sensors are checked to detect whether signals have been emitted by other components;

3) timing constraints of the output transitions of the current location of the timed automaton are evaluated with the value of T. If one of these conditions is satisfied, then one of the firable transitions is taken.

The program semantics of $\mathcal{A}$ depends on two parameters Δ_P, Δ_L and is denoted by $[\![\mathcal{A}]\!]^{\mathsf{Prog}}_{\Delta_L,\Delta_P}$. These two parameters represent the performance of the platform. More precisely, they are defined as follows:

1) the clock register is incremented every Δ_P time units;

2) the time necessary to process one cycle of instructions is less or equal than Δ_L.

The fundamental result that links the two previously introduced semantics is:

THEOREM 9.12 (Simulation [DEW 05, theorem 4]) *Let $\mathcal{A}$ be a timed automaton and $\Delta, \Delta_P, \Delta_L \in \mathbb{Q}_{>0}$ three rational parameters. Then if the inequality $3\Delta_L + 4\Delta_P \leq \Delta$ is satisfied, the associated transition systems verify:*

$$[\![\mathcal{A}]\!]^{\textsf{\textit{Prog}}}_{\Delta_L,\Delta_P} \sqsubseteq [\![\mathcal{A}]\!]^{\textsf{\textit{AASAP}}}_{\Delta}$$

As for any positive Δ, it is always possible to find parameters values of Δ_L and Δ_P verifying the above inequality, this result establishes that if the AASAP semantics is correct for some value of Δ, then there exists a correct implementation of $\mathcal{A}$.

Finally, we present a last result, stating that the AASAP semantics can be simulated by a syntactic transformation of timed automata, involving a parameter.

THEOREM 9.13 ([DEW 05, theorem 8]) *Let $\mathcal{A}$ be a timed automaton and $\Delta \in \mathbb{Q}_{>0}$. It is possible to build a timed automaton $\mathcal{F}(\mathcal{A}, \Delta)$ such that:*

$$[\![\mathcal{A}]\!]^{\textsf{\textit{AASAP}}}_{\Delta} \sqsubseteq [\![\mathcal{F}(\mathcal{A},\Delta)]\!] \text{ and } [\![\mathcal{F}(\mathcal{A},\Delta)]\!] \sqsubseteq [\![\mathcal{A}]\!]^{\textsf{\textit{AASAP}}}_{\Delta}$$

The main operation involved in the definition of $\mathcal{F}(\mathcal{A}, \Delta)$ is the enlargment of guards and invariants by Δ. Intuitively, this consists in replacing a constraint of the form $x \in [a, b]$ by the constraint $x \in [a - \Delta, b + \Delta]$. As the other transformations involved in $\mathcal{F}(\mathcal{A}, \Delta)$ are less important, we consider in the chapter a parametric semantics of timed automata:

DEFINITION 9.14 (Enlarged semantics) *Let $\mathcal{A}$ be a timed automaton. We define the enlarged semantics of $\mathcal{A}$ as a parametric semantics depending on a rational parameter Δ. It is defined by the timed transition system associated with the timed automaton obtained from $\mathcal{A}$ by enlarging all guards and invariants of Δ, and denoted by $[\![\mathcal{A}]\!]_{\Delta}$.*

The problem of interest is, to decide whether a positive value of Δ exists for which the enlarged semantics meets a specification given by a universal property on executions.

9.4.3. *Related work*

Recently, several works have proposed approaches to ensure implementability of timed models. In this section, we present one of them, which relies on a parametric enlargment of timed automata. We present a short survey of related works:

– this approach using the AASAP semantics contrasts with a modeling-based solution proposed in [ALT 05], where the behavior of the platform is modelled as a timed automaton. While this approach offers a very rich framework to model the platform, it suffers from two drawbacks. First, it does not verify the "faster is better property" (if a model is implementable on an implementation, then it should also be the case on any "faster" implemention). Second, verification problems are difficult and the approach of [ALT 05] does not offer decidability results;

– other works has introduced other notions of "robustness" in order to relax the mathematical idealization of the semantics of timed automata [GUP 97, OUA 03, BAI 07b, BAI 07a]. Those approaches are different from ours, since they roughly consist of dropping "isolated" or "unlikely" executions;

– these results should also be compared with tools related to code generation from timed automata. The TIMES tool [AMN 03] automatically generates executable code from timed automata. However, it does not take into account the imprecisions, as it relies on the perfect synchrony paradigm. Henzinger *et al.* proposed in [HEN 03] a model for embedded systems, called GIOTTO, which can be seen as an intermediary step beween timed automata and real platforms.

9.4.4. *Decidability result for safety properties*

We consider here a safety property, which is given by a set of locations of the timed automaton that should be avoided and is denoted Bad. Thus, in the enlarged semantics, the problem we want to solve is the following: does there exist a positive value of Δ such that $Reach([\![\mathcal{A}]\!]_\Delta)$ does not intersect Bad?

This problem has been solved in [PUR 00, DEW 08], under some assumptions on timed automata that we recall here. In this subsection, we assume that the timed automata we consider satisfy the following requirements:

– all guards and invariants involve non-strict inequalities;

– all the clocks are always bounded;

– all the cycles of the region graph are *progress cycles*, i.e. do reset all clocks to zero.

The first hypothesis does not change the expressive power of the model under the enlarged semantics. The second hypothesis is not really restrictive since every timed automaton can be transformed into such a bounded timed automaton (see for example [BEH 01]). Note that it entails that any time-divergent path contains an infinite number of action transitions. As mentioned in [DEW 08], the third requirement is less restrictive than classical strong-non-Zenoness assumptions.

The proof of the decidability result relies on the following proposition:

PROPOSITION 9.15 *Let $\mathcal{A}$ be a timed automaton, and* Bad *be a set of locations. We have:*

$$\exists \Delta > 0 \mid Reach(\llbracket \mathcal{A} \rrbracket_\Delta) \cap \textsf{Bad} = \emptyset \iff \left(\bigcap_{\Delta > 0} Reach(\llbracket \mathcal{A} \rrbracket_\Delta) \right) \cap \textsf{Bad} = \emptyset$$

Intuitively, this property follows from the fact that two subsets of $\mathbb{R}^n_{\geq 0}$, defined by zones[4] with integral constraints, and whose intersection is empty, are at distance at least $\frac{1}{n}$. In addition, as all guards are non-strict, we can prove that all sets involved in the proposition verify these conditions.

As a consequence, the problem is reduced to the computation of the set $Reach^*(\llbracket \mathcal{A} \rrbracket)$ defined as $\bigcap_{\Delta>0} Reach(\llbracket \mathcal{A} \rrbracket_\Delta)$. This set exactly contains configurations that are reachable for any *positive* value of the parameter.

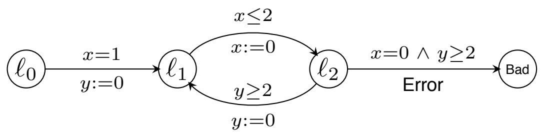


Figure 9.10. *A timed automaton $\mathcal{A}$*

This set can differ from the set $Reach(\llbracket \mathcal{A} \rrbracket)$. As an illustration, consider the timed automaton depicted in Figure 9.10, and where ρ is the simple cycle around locations ℓ_1 and ℓ_2. The two corresponding sets are depicted on Figure 9.11, together with the sets of reachable configurations after one and two executions of cycle ρ. In this example, the guard enlargment allows the first transition of the cycle (from ℓ_1 to ℓ_2) to be fired a bit after the constraint $x = 2$ is satisfied and the second transition to be fired a bit before the constraint $y = 2$ is satisfied. As a consequence, this allows location ℓ_1 to be reached, after one loop, with a clock valuation of $v(x) = 1 - 3\Delta$ and $v(y) = 0$ (see Figure 9.10). This slight perturbation can be accumulated by iterating the cycle, after a second loop, a clock valuation of $v(x) = 1 - 5\Delta$ and $v(y) = 0$ is obtained (see Figure 9.10). As a consequence, whatever how small Δ is, any valuation of the set $Reach^*(\llbracket \mathcal{A} \rrbracket)$ can be reached after a sufficient number of iterations of the cycle.

4. A zone is defined by a conjunction of lower and upper bounds on clocks and clock differences.

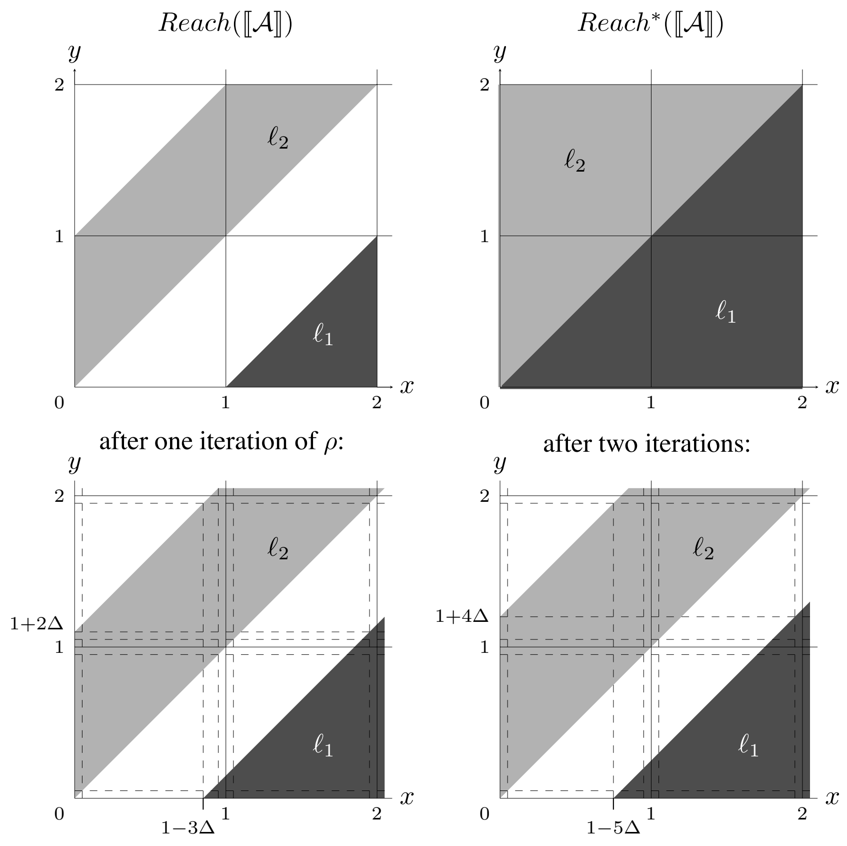


Figure 9.11. *Differences between $Reach([\![\mathcal{A}]\!])$ and $Reach^*([\![\mathcal{A}]\!])$, and intermediate computations of the reachable configurations in $[\![\mathcal{A}]\!]_\Delta$.*

Algorithm 5 can be used to compute the set $Reach^*([\![\mathcal{A}]\!])$, which was proposed in [PUR 00]. This algorithm relies on the region graph construction presented in section 9.2. In its presentation, notation $Reach(\mathcal{R}(\mathcal{A}), J^*)$ denotes the set of reachable states from J^* in $\mathcal{R}(\mathcal{A})$. In addition, given a region R of a timed automaton, its topological closure is denoted $\overline{R}$, and this notation is extended to sets of regions.

The proof of correction of the algorithm is long and technical, and thus we only briefly describe it. The proof proceeds by double inclusion. To prove the soundness of the algorithm, i.e. the inclusion $J^* \subseteq Reach^*([\![\mathcal{A}]\!])$, the authors show a key lemma stating that two configurations belonging to the topological closure of a same strongly connected component can reach each other, for any postive value of Δ. For the completeness, the authors prove a bound between the distance of a run in the enlarged

```
input  : a timed automaton $\mathcal{A}$
output: the enlarged reachability set $Reach^*(\llbracket\mathcal{A}\rrbracket)$
1 Compute the region graph $\mathcal{R}(\mathcal{A}) = (\Gamma, \gamma_0, \rightarrow)$
2 Compute the set $\mathcal{S}$ of strongly connected components of $\mathcal{R}(\mathcal{A})$
3 $J^* := \{\gamma_0\}$
4 $J^* := Reach(\mathcal{R}(\mathcal{A}), J^*)$
5 while $\exists S \in \mathcal{S}$ such that $S \not\subseteq J^*$ and $J^* \cap \overline{S} \neq \emptyset$ do
6     $J^* := J^* \cup \overline{S}$
7     $J^* := Reach(\mathcal{R}(\mathcal{A}), J^*)$
8 Return $J^*$;
```

Algorithm 5: Computation of the set $Reach^*(\mathcal{A})$

semantics and a run in the standard semantics, depending on the parameter Δ. Using the fact that long paths will necessarily go through strongly connected components, they prove that this bound vanishes when $\Delta \to 0$. Finally, we obtain the following theorem.

THEOREM 9.16 *Let $\mathcal{A}$ be a timed automaton and* Bad *be a subset of locations of $\mathcal{A}$. One can decide whether there exists a positive value of parameter Δ such that $Reach(\llbracket\mathcal{A}\rrbracket_\Delta) \cap \mathsf{Bad} = \emptyset$. In addition, this problem is PSPACE-complete.*

Recall that the emptiness problem for timed automata is already PSPACE-complete (theorem 9.2).

9.4.5. *Other results and current challenges*

This semantical approach to the robustness and implementability of timed automata has been extended in several works in the recent years. We describe here some of these results.

Symbolic computations

First, as we mentionned in the introduction, the decidability results based on the region graph construction do not yield efficient algorithms. In [DAW 06], a symbolic algorithm is proposed for the computation of $Reach^*(\mathcal{A})$. This algorithm uses zones as a symbolic representation of clock valuations. The computation of the strongly connected components of the region graph is replaced by a notion of stable zone of a cycle, which corresponds intuitively to the set of valuations that are connected through this cycle, regardless how small the perturbation is. The authors prove that the stable zone can be computed as a greatest fixpoint. However, their approach only handles flat timed automata, as the computation of the stable zone must be done for each cycle of the system. Flat means here that the system does not contain nested cycles.

Another perturbation

In the seminal paper by Puri [PUR 00], the notion of guard enlargment is introduced, and also a notion of drift of clocks. This is formalized, given a positive rational number $\varepsilon > 0$, by modifying the semantics of the timed automaton by allowing clocks to have a rate in the interval $[1 - \varepsilon, 1 + \varepsilon]$. As a consequence, clocks may not evolve at the same speed.

This notion of perturbation is very attractive, as it intuitively corresponds to actual flaws of real systems. It turns out to be equivalent to guard enlargment. More formally, it was proved in [PUR 00, DEW 08] that a similar definition of the set $Reach^*(\mathcal{A})$ for clock drift yields the same set. The two perturbations can be combined and this still yields the same set of configurations (see [DEW 08] for a detailed proof).

Robust verification

The results presented so far concern safety properties. In [BOU 06, BOU 08b], the robust verification is considered for other properties such as temporal properties. First, it is proven that the robust model-checking of LTL properties is PSPACE-complete, as for standard model-checking. Second, an expressive fragment of the timed logic MTL is proposed, for which the robust model-checking is also decidable. This fragment encompasses the logic LTL, the bounded fragment of MTL, and can express interesting properties such as bounded response-time. Again, the complexity obtained is the same as for standard model-checking.

Quantitative robustness

Using the previous results, we can decide whether there a positive perturbation exists for which the property is satisfied. In [JAU 11], a more difficult problem is tackled, which consists of computing the largest value of the parameter for the system to meet the specification. The authors propose a parametric extension of the symbolic algorithm introduced in [DAW 06], which allows the set of reachable states in the timed automaton to be computed, for all values of the parameter, which corresponds to the parametric reachability set. However, as for [DAW 06], the algorithm only applies to flat timed automata.

Current challenges

This semantical approach of robustness opens promising perspectives for research. Among them, we can mention the problem of robust control. This problem consists of synthesizing a controller, for instance from a timed game specification, which is robust, i.e. such that a positive enlargment exists for which the enlarged controller still ensures that the system under control is correct. This problem has been investigated in [CHA 08], but only for a fixed value of the enlargment.

9.5. Conclusion

Real-time aspects are central to the design of many applications, such as embedded systems. We have presented in this chapter how verification techniques can be extended to timed systems. Therefore, we have focused on the notion of region, which consists of gathering configurations of the system in equivalence classes, yielding a quotient system on which the property can be checked.

In addition to region constructions, existing tools for the verification of timed systems may also use other symbolic techniques such as zones (see Uppaal [BEH 04], Romeo [GAR 05]). Recently, several case studies have proven that these tools have reached a level of maturity that allows them to tackle relevant industrial problems.

Current works are investigating extensions to these techniques to more complex models. We have presented recent works related to the implementability of timed systems. Among other directions, let us mention two-player games, which are useful for controller synthesis problems; the extensions to weighted systems which allow variables with more complex dynamics than clock variables to modeled; and probabilistic systems, used for instance to model communication protocols.

9.6. Bibliography

[ABD 01] ABDULLA P. A., NYLÉN A., "Timed Petri nets and BQOs", *in Proc. 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, vol. 2075 of *LNCS*, Springer, p. 53–70, 2001.

[ABD 07] ABDULLA P. A., MAHATA P., MAYR R., "Dense-timed Petri nets: checking Zenoness, token liveness and boundedness", *Logical Methods in Computer Science*, vol. 3, num. 1, p. 1–61, 2007.

[ALT 05] ALTISEN K., TRIPAKIS S., "Implementation of timed automata: an issue of semantics or modeling?", *in Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, vol. 3829 of *LNCS*, Springer, p. 273-288, 2005.

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

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

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

[AMN 03] AMNELL T., FERSMAN E., MOKRUSHIN L., PETTERSSON P., YI W., "TIMES: A tool for schedulability analysis and code generation of real-time systems", *in Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, vol. 2791 of *LNCS*, Springer, p. 60–72, 2003.

[BAI 07a] BAIER C., BERTRAND N., BOUYER P., BRIHAYE TH., GRÖSSER M., Almost-sure model checking of infinite paths in one-clock timed automata, Research Report num. LSV-07-29, ENS Cachan, France, 2007.

[BAI 07b] BAIER C., BERTRAND N., BOUYER P., BRIHAYE TH., GRÖSSER M., "Probabilistic and topological semantics for timed automata", *in Proc. 27th Conf. Found. Softw. Tech. & Theor. Comp. Sci. (FSTTCS'07)*, vol. 4855 of *LNCS*, Springer, p. 179–191, 2007.

[BEH 01] BEHRMANN G., FEHNKER A., HUNE TH., LARSEN K. G., PETTERSSON P., ROMIJN J., VAANDRAGER F., "Minimum-cost reachability for priced timed automata", *in Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, vol. 2034 of *LNCS*, Springer, p. 147–161, 2001.

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

[BER 83] BERTHOMIEU B., MENASCHE M., "An enumerative approach for analyzing time Petri nets", *in Proc. 9th IFIP Congress*, Elsevier Science Publishers, p. 41–46, 1983.

[BER 91] BERTHOMIEU B., DIAZ M., "Modeling and verification of time dependent systems using time Petri nets", *IEEE Transactions in Software Engineering*, vol. 17, num. 3, p. 259–273, 1991.

[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.

[BER 03] BERTHOMIEU B., VERNADAT F., "State Class Constructions for Branching Analysis of Time Petri Nets", *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, vol. 2619 of *Lecture Notes in Computer Science*, Springer, p. 442–457, 2003.

[BÉR 09] BÉRARD B., HADDAD S., "Interrupt timed automata", *in Proc. 12th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'09)*, vol. 5504 of *LNCS*, Springer, p. 197-211, 2009.

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

[BOU 06] BOUYER P., MARKEY N., REYNIER P.-A., "Robust model-checking of linear-time properties in timed automata", *in Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, vol. 3887 of *LNCS*, Valdivia, Chile, Springer, p. 238-249, 2006.

[BOU 08a] BOUYER P., HADDAD S., REYNIER P.-A., "Timed Petri nets and timed automata: On the discriminating power of zeno sequences", *Inf. Comput.*, vol. 206, num. 1, p. 73-107, 2008.

[BOU 08b] BOUYER P., MARKEY N., REYNIER P.-A., "Robust analysis of timed automata *via* channel machines", *in Proc. 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, vol. 4962 of *LNCS*, Springer, p. 157-171, 2008.

[BRO 88] BROWNE M. C., CLARKE E. M., GRUMBERG O., “Characterizing finite Kripke structures in propositional temporal logic.”, *Theoretical Computer Science*, vol. 59, p. 115-131, 1988.

[CAS 02] CASSEZ F., HENZINGER TH. A., RASKIN J.-F., “A comparison of control problems for timed and hybrid systems”, *in Proc. 5th International Workshop on Hybrid Systems: Computation and Control (HSCC’02)*, vol. 2289 of *LNCS*, Springer, p. 134–148, 2002.

[CHA 08] CHATTERJEE K., HENZINGER T. A., PRABHU V. S., “Timed parity games: complexity and robustness”, *in Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, vol. 5215 of *LNCS*, Springer, p. 124-140, 2008.

[DAW 96] DAWS C., OLIVERO A., TRIPAKIS S., YOVINE S., “The tool Kronos”, *in Proc. Hybrid Systems III: Verification and Control (1995)*, vol. 1066 of *LNCS*, Springer, p. 208–219, 1996.

[DAW 06] DAWS C., KORDY P., “Symbolic robustness analysis of timed automata.”, *in Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS’06)*, vol. 4202 of *LNCS*, Springer, p. 143-155, 2006.

[DEW 05] DE WULF M., DOYEN L., RASKIN J.-F., “Almost ASAP semantics: from timed models to timed implementations”, *Formal Aspects of Computing*, vol. 17, num. 3, p. 319-341, 2005.

[DEW 08] DE WULF M., DOYEN L., MARKEY N., RASKIN J.-F., “Robust safety of timed automata”, *Formal Methods in System Design*, vol. 33, num. 1-3, p. 45-84, Kluwer Academic Publishers, 2008.

[GAR 05] GARDEY G., LIME D., MAGNIN M., ROUX O. H., “Romeo: A tool for analyzing time Petri nets”, *in Proc. 17th International Conference on Computer Aided Verification (CAV’05)*, vol. 3576 of *LNCS*, Springer, p. 418–423, 2005.

[GUP 97] GUPTA V., HENZINGER TH. A., JAGADEESAN R., “Robust timed automata”, *in Proc. International Workshop on Hybrid and Real-Time Systems (HART’97)*, vol. 1201 of *LNCS*, Springer, p. 331–345, 1997.

[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 03] HENZINGER TH. A., HOROWITZ B., KIRSCH C. M., “GIOTTO: A time-triggered language for embedded programming”, *in Proc. of the IEEE*, vol. 91, num. 1, p. 84–99, 2003.

[HIG 52] HIGMAN G., “Ordering by divisibility in abstract algebras”, *in Proc. London Mathematical Society*, vol. 2, p. 326–336, 1952.

[JAU 11] JAUBERT R., REYNIER P.-A., “Quantitative robustness analysis of flat timed automata”, *in Proc. 14th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS’11)*, LNCS, Springer, 2011, To appear.

[KAR 69] KARP R. M., MILLER R. E., “Parallel program schemata.”, *Journal of Computer and System Sciences*, vol. 3, num. 2, p. 147-195, 1969.

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

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

[MAH 05] MAHATA P., Model checking parameterized timed systems, PhD thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2005.

[OUA 03] OUAKNINE J., WORRELL J. B., "Revisiting digitization, robustness and decidability for timed automata", *in Proc. 18th Annual Symposium on Logic in Computer Science (LICS'03)*, IEEE Computer Society Press, 2003.

[OUA 04] OUAKNINE J., WORRELL J. B., "On the language inclusion problem for timed automata: closing a decidability gap", *in 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. B., "On the decidability of metric temporal logic", *in Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05)*, IEEE Computer Society Press, p. 188–197, 2005.

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

[PUR 00] PURI A., "Dynamical properties of timed automata", *Discrete Event Dynamic Systems*, vol. 10, num. 1-2, p. 87–113, 2000.