1

Parametric Timed Automata

In this chapter, we present the formalisms used throughout this book. In particular, we present timed automata [ALU 94], a powerful modeling formalism for real-time systems. Since this book focuses on synthesizing values for timing parameters of a system, guaranteeing a good behavior, we will also use a parametric extension of timed automata, namely *parametric timed automata* [ALU 93c]. This chapter presents their syntax and semantics, and more generally all the necessary formalisms to understand the rest of this book. Any reader who is not particularly interested in theory can skip directly to Chapter 2, and return to Chapter 1 when needed.

Outline of the chapter

We describe clocks, parameters and constraints on the clocks and parameters in section 1.1 and labeled transition system in section 1.2. We then introduce the syntax and semantics of timed automata in section 1.3, and parametric timed automata in section 1.4. Related works, including representation of time, and formalisms related to timed automata, are discussed in section 1.5.

1.1. Constraints on clocks and parameters

1.1.1. *Clocks*

Throughout this book, we assume a fixed set $X = \{x_1, \ldots, x_H\}$ of *clocks*. A *clock* is a variable x_i with value in $\mathbb{R}_+$, which denotes the set of non-negative real numbers. All clocks evolve linearly at the same rate. We define a *clock valuation* as a function $w\colon X \to \mathbb{R}_+$ assigning a non-negative real value to each clock variable. We will often identify a valuation w with the point $(w(x_1), \ldots, w(x_H))$. Given a constant $d \in \mathbb{R}_+$, we use $X + d$ to denote the set $\{x_1 + d, \ldots, x_H + d\}$. Similarly, we write $w + d$ to denote the valuation such that $(w + d)(x) = w(x) + d$ for all $x \in X$.

1.1.2. *Parameters*

Throughout this book, we assume a fixed set $P = \{p_1, \ldots, p_M\}$ of *parameters*, that is unknown constants. A *parameter valuation* π is a function π: $P \to \mathbb{R}_+$ assigning a non-negative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}_+)^M$. We will often identify a valuation π with the point $(\pi(p_1), \ldots, \pi(p_M))$.

1.1.3. *Constraints*

We define constraints here as a set of linear inequalities.

1.1.3.1. *Syntax of constraints*

DEFINITION 1.1.– *Let V be a set of variables of the form $V = \{v_1, \ldots, v_N\}$. A* linear inequality on the variables of V *is an inequality $e \prec e'$, where $\prec \in \{<, \leq\}$ and e, e' are two linear terms of the form:*

$$\sum_{1 \leq i \leq N} \alpha_i v_i + d$$

where $v_i \in V$, $\alpha_i \in \mathbb{R}_+$, for $1 \leq i \leq N$ and $d \in \mathbb{R}_+$.

Note that we define the coefficients of the linear inequalities as *positive reals*. It would be equivalent to define them as *positive rationals*, since we consider only linear inequalities. Both definitions are found in the literature; we suppose here that, since we are addressing the problem of the verification of real-time systems, we consider real valued constants.

We assume in the following that all inequalities are linear, and we will simply refer to linear inequalities as *inequalities*.

DEFINITION 1.2.– *Let V be a set of variables of the form $V = \{v_1, \ldots, v_N\}$. Given an inequality J on the variables of V of the form $e < e'$ (respectively, $e \leq e'$), the* negation *of J, denoted by $\neg J$, is the linear inequality $e' \leq e$ (respectively, $e' < e$).*

DEFINITION 1.3.– *Let V be a set of variables of the form $V = \{v_1, \ldots, v_N\}$. A* convex linear constraint on the variables of V *is a conjunction of inequalities on the variables of V.*

We assume in the following that all constraints are both convex and linear, and we will simply refer to convex linear constraints as *constraints*.

DEFINITION 1.4.– *An* inequality on the clocks *is an inequality on the set of clocks X. A* constraint on the clocks *is a constraint on the set of clocks X.*

DEFINITION 1.5.– *An* inequality on the parameters *is an inequality on the set of parameters P. A* constraint on the parameters *is a constraint on the set of parameters P.*

DEFINITION 1.6.– *An* inequality on the clocks and the parameters *is an inequality on $X \cup P$. A* constraint on the clocks and the parameters *is a constraint on $X \cup P$.*

Throughout this book, we denote by $\mathcal{L}(X)$ the set of all constraints on the clocks, by $\mathcal{L}(P)$ the set of all constraints on the parameters and by $\mathcal{L}(X \cup P)$ the set of all constraints on the clocks and the parameters.

In the following, the letter J will denote an inequality on the parameters, the letter D will denote a constraint on the clocks, the letter K will denote a constraint on the parameters, and the letter C will denote a constraint on the clocks and the parameters.

1.1.3.2. *Semantics of constraints*

Given a constraint D on the clocks and a clock valuation w, $D[w]$ denotes the expression obtained by replacing each clock x in D with $w(x)$. A clock valuation w *satisfies* constraint D (denoted by $w \models D$) if $D[w]$ evaluates to true.

Given a parameter valuation π and a constraint C on the clocks and the parameters, $C[\pi]$ denotes the constraint on the clocks obtained by replacing each parameter p in C with $\pi(p)$. Likewise, given a clock valuation w, $C[\pi][w]$ denotes the expression obtained by replacing each clock x in $C[\pi]$ with $w(x)$. We say that a parameter valuation π *satisfies* a constraint C, denoted by $\pi \models C$, if the set of clock valuations that satisfy $C[\pi]$ is non-empty. We use the notation $<w, \pi> \models C$ to indicate that $C[\pi][w]$ evaluates to true.

A convex linear constraint on the clocks and the parameters can also be interpreted as a set of points in the space $\mathbb{R}^{M+H}$, more precisely as a convex polyhedron. We will use these notions synonymously. In this geometric context, a valuation w satisfying a constraint C is equivalent to the polyhedron C containing the corresponding point w, written as $w \in C$. For a partial valuation w (i.e. a point of a subspace of C), we write $w \in C$ if and only if w is contained in the projection of C on the variables of w.

Given two constraints C_1 and C_2 on the clocks and the parameters, we say that C_1 is *included in* C_2, denoted by $C_1 \subseteq C_2$, if $\forall w, \pi : <w, \pi> \models C_1 \Rightarrow <w, \pi> \models C_2$. We have that $C_1 = C_2$ if and only if $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$.

Similarly to the semantics of constraints on the clocks and the parameters, we say that a parameter valuation π *satisfies* a constraint K on the parameters, denoted by $\pi \models K$, if the expression obtained by replacing each parameter p in K with $\pi(p)$ evaluates to true. Given two constraints K_1 and K_2 on the parameters, we say that K_1 is *included in* K_2, denoted by $K_1 \subseteq K_2$, if $\forall \pi : \pi \models K_1 \Rightarrow \pi \models K_2$. We have

that $K_1 = K_2$ if and only if $K_1 \subseteq K_2$ and $K_2 \subseteq K_1$. We will consider `true` as a constraint on the parameters, corresponding to the set of all possible values for P.

Given a constraint C on the clocks and the parameters, we denote by $C{\downarrow_P}$ the constraint on the parameters obtained by projecting C onto the set of parameters, that is after elimination of the clock variables. Formally:

$$C{\downarrow_P} = \{\pi \mid \exists w : <w, \pi> \models C\}.$$

Sometimes we will refer to a variable domain X', which is obtained by renaming the variables in X. Explicit renaming of variables is denoted by the substitution operation. Given a constraint C on the clocks and the parameters, we denote by $C_{[X \leftarrow X']}$ the constraint obtained by replacing in C the variables of X by the variables of X'.

We define the *time elapsing* of C, denoted by $C{\uparrow}$, as the constraint over X and P obtained from C by delaying an arbitrary amount of time. Note that, of course, only clocks can evolve; parameters are unknown constants and therefore remain constant. Formally:

$$C{\uparrow} = \Big((C \wedge X' = X + d){\downarrow_{X' \cup P}}\Big)_{[X' \leftarrow X]}$$

where d is a new parameter with values in $\mathbb{R}_+$ and X' is a renamed set of clocks. The inner part of the expression adds a delay d to all clocks; the projection onto $X' \cup P$ eliminates the original set of clocks X, as well as the variable d; the outer part of the expression renames clocks X' with X.

1.2. Labeled transition systems

We now introduce labeled transition systems, which will be used later in this section to represent the semantics of timed automata.

DEFINITION 1.7.– *A* labeled transition system *over a set of symbols* Σ *is a triple* $\mathrm{L} = (S, S_0, \Rightarrow)$*, with* S *a set of* states*,* $S_0 \subset S$ *a set of* initial states *and* $\Rightarrow \in S \times \Sigma \times S$ *a* transition relation*. We write* $s \overset{a}{\Rightarrow} s'$ *for* $(s, a, s') \in \Rightarrow$*. A* run *(of length* m*) of* L *is a finite alternating sequence of states* $s_i \in S$ *and symbols* $a_i \in \Sigma$ *of the form* $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$*, where* $s_0 \in S_0$*. A state* s_i *is* reachable *if it belongs to some run* r*.*

1.3. Timed automata

Timed automata are an extension of standard finite-state automata allowing the use of clocks, that is real-valued variables increasing linearly at the same rate. Such clocks

can be compared with constants in constraints that allow us (or not) to stay in a location ("invariants") or to take a transition ("guards"). At each transition, it is possible to reset some of the clocks of the system. This formalism allows the parallel composition of several timed automata, which behave like a single one, and thus provides the designer with a powerful and intuitive way to represent timed systems. It is important to note that the formalism of timed automata is very sensitive to the size of the automata and the number of automata in parallel, thus often leading to the state-space explosion problem. However, powerful tools, such as the UPPAAL [LAR 97] model checker, have been implemented, allowing designers to model and verify very efficiently timed systems modeled by timed automata.

1.3.1. *Syntax*

DEFINITION 1.8.– *A* timed automaton $\mathcal{A}$ *is a* 6*-tuple of the form* $\mathcal{A} = (\Sigma, Q, q_0, X, I, \rightarrow)$*, where:*

- Σ *is a finite set of* actions*,*
- Q *is a finite set of* locations*,*
- $q_0 \in Q$ *is the* initial location*,*
- X *is a set of* clocks*,*
- $I : Q \rightarrow \mathcal{L}(X)$ *is the* invariant*, assigning to every* $q \in Q$ *a constraint* $I(q)$ *on the clocks and*
- $\rightarrow$ *is a transition relation consisting of elements of the form* (q, g, a, ρ, q')*, also denoted by* $q \overset{g,a,\rho}{\rightarrow} q'$*, where* $q, q' \in Q$*,* $a \in \Sigma$*,* $\rho \subseteq X$ *is a set of clock variables to be* reset *by the transition, and* $g \in \mathcal{L}(X)$ *is the* guard *of the transition.*

Note that we use a more permissive definition of the constraints used in guards and invariants than in the original definition of timed automata (see [ALU 94]). Indeed, we allow the use of conjunctions of any linear inequalities on the clocks, whereas the original definition usually considers conjunctions of comparisons of a single clock with a constant. This more permissive definition usually has an impact on the decidability (the addition of clock values within a constraint leads to undecidability [ALU 94]), but this has no impact in this book, mainly because of the use of *parametric* timed automata, where the parameters bring themselves undecidability in the general case. Furthermore, many tools for (parametric) timed automata allow more permissive definitions than the original one.

Timed automata are often extended in practice with *discrete variables*, which can be used in guards and transitions, updated within the transitions, and sometimes even used as a factor for clocks. However, in most cases, they represent only syntactic

sugar for the discrete space (i.e. locations). As a result, we will not use them in any theoretical part of this book. Note, nevertheless, that many tools for (parametric) timed automata allow the use of such discrete variables, and some of the case studies contained here also use them.

The graphical representation of a timed automaton $\mathcal{A}$ is an oriented graph where vertices correspond to locations, and edges correspond to actions of $\mathcal{A}$. We follow the following conventions for the graphical representation of timed automata: locations are represented by nodes, above of which the invariant of the location is written; transitions are represented by arcs from one location to another location, labeled by the associated guard, the action name and the set of clocks to be reset (guards and invariants equal to `true` are omitted). The initial location is represented here using a double circle.

EXAMPLE 1.1.– We give in Figure 1.1 an example of a timed automaton containing four locations (viz. q_0, q_1, q_2 and q_3), three actions (viz. a, b and c) and two clocks (viz. x_1 and x_2). The initial location is q_0.

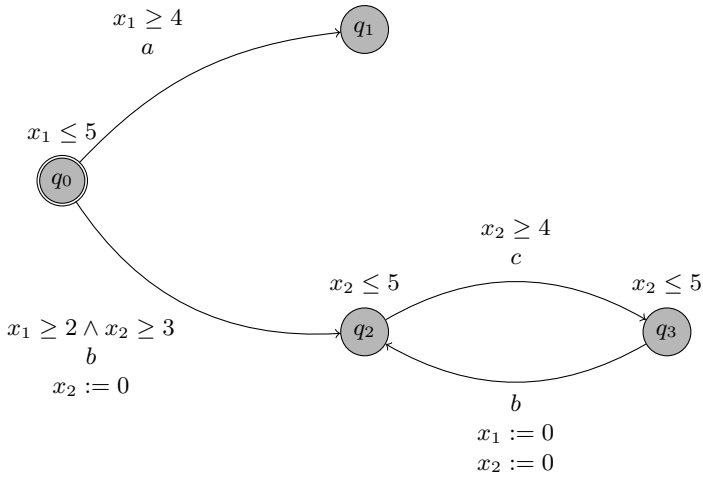


Figure 1.1. *An example of a timed automaton*

In this timed automaton, q_0 has invariant $x_1 \leq 5$, q_1 has invariant `true` and both q_2 and q_3 have invariant $x_2 \leq 5$. The transition from q_0 to q_1 has guard $x_1 \geq 4$ through action a; no clock is reset. The transition from q_0 to q_2 has guard $x_1 \geq 2 \wedge x_2 \geq 3$ through action b, and resets clock x_2. The transitions between q_2 and q_3 can be explained similarly.

1.3.2. *Semantics*

The semantics of timed automata is given under the form of a labeled transition system, where states are pairs made by a location and a valuation for each clock.

DEFINITION 1.9.– *Let $\mathcal{A} = (\Sigma, Q, q_0, X, I, \rightarrow)$ be a timed automaton. The* concrete semantics *of $\mathcal{A}$ is the labeled transition system $(S, S_0, \Rightarrow)$ over Σ, where*

$$S = \{(q, w) \in Q \times (X \to \mathbb{R}_+) \mid w \models I(q)\},$$
$$S_0 = \{(q_0, w) \mid w \models I(q_0) \wedge w = (w_0, \ldots, w_0) \text{ for some } w_0 \in \mathbb{R}_+\}$$

and the transition predicate $\Rightarrow$ is specified by the following three rules. For all $(q, w), (q', w') \in S, d \geq 0$ and $a \in \Sigma$,

- $(q, w) \xrightarrow{a} (q', w')$ *if* $\exists g, \rho : q \xrightarrow{g, a, \rho} q'$ *and* $w \models g$ *and* $w' = \rho(w)$*;*
- $(q, w) \xrightarrow{d} (q', w')$ *if* $q' = q$ *and* $w' = w + d$*;*
- $(q, w) \overset{a}{\Rightarrow} (q', w')$ *if* $\exists d, w'' : (q, w) \xrightarrow{a} (q', w'') \xrightarrow{d} (q', w')$.

We consider with the definition of S_0 that all clocks are initially set to 0, or have evolved linearly in the bounds given by $I(q_0)$. A state (respectively, run) in the concrete semantics will be referred to as a *concrete state* (respectively, *concrete run*).

A concrete run is represented under the form of a branch where states are shown within nodes containing the name of the location and the value of each of the clocks, and transitions are shown using edges labeled with the name of the action.

EXAMPLE 1.2.– Consider again the timed automaton $\mathcal{A}$ of example 1.1. Then, Figure 1.2 shows an example of a concrete run for $\mathcal{A}$.

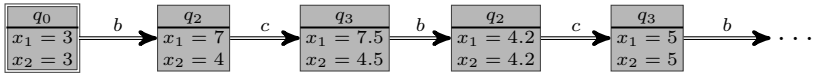


Figure 1.2. *Example of a concrete run for a timed automaton*

This run is obtained as follows: we start from the initial location q_0 where both clocks have evolved during three time units. Then, we take action b, reset x_2 and spend four time units in q_2. Then, we take action c, and spend 0.5 time unit in q_3. Then, we take action b, reset both clocks and spend 4.2 time units in q_2. Then, we take action c, and spend 0.8 time units in q_3, and so on.

The power of timed automata relies on the fact that one can construct a finite partition of the infinite space of clock valuations. In particular, this construction is suitable to perform reachability analysis. The main theoretical advantage of timed

automata relies in its decidability results. In particular, it has been shown that the reachability of a state is decidable. Moreover, various *timed* temporal logics (e.g. [ALU 93a]) have been designed and various decidability results have been shown (e.g. [ALU 93a, HUN 02, WAN 03, FIN 06, BAI 09]).

1.3.2.1. *Traces*

We now introduce the notion of *trace*, that abstracts part of a system's behavior. In the literature, we usually consider either a state-based approach or an action-based approach (see, e.g. [BAI 08]). We consider here a combined state- and action-based approach: a trace is an alternating sequence of locations and actions. Note that, for a deterministic timed automaton, that is a timed automaton such that there is at most one transition leaving a given location with a given action, there is an equivalence between the state-based, the action-based and the combined approaches (because of the unicity of the initial location). This can be the case for hardware verification when one models circuits at the gate level. Indeed, when we model each gate of the circuit with a different timed automaton, where each location corresponds to a different value of the input and output signals of the gate, and each transition corresponds to a rise or a fall of a signal of the global system, then the composition of the timed automata modeling each gate is deterministic. In that case, if we consider a sequence of locations, it is possible to retrieve the corresponding sequence of actions (from a given initial location), and conversely.

We define more formally the notion of trace in the following definition.

DEFINITION 1.10.– *Given a timed automaton $\mathcal{A}$ and a concrete run r of $\mathcal{A}$ of the form $(q_0, w_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, w_m)$, the* trace associated with r *is the alternating sequence of locations and actions $q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$. We say that location q_i, for $1 \leq i \leq m$, belongs to the trace.*

A trace is built from a run by removing the valuation of the clocks, and therefore can be seen as a *time-abstract run*. We show traces under a graphical form using boxed nodes labeled with locations and double arrows labeled with actions.

EXAMPLE 1.3.– The trace associated with the concrete run of example 1.2 is shown in Figure 1.3.

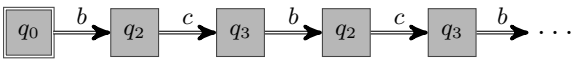


Figure 1.3. *Example of a trace associated with a concrete run for a timed automaton*

We define below the notion of acyclic trace as a trace that never passes twice by the same location, that is a trace whose locations are all different.

DEFINITION 1.11.– *Given a trace* $T = q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$*,* T *is said to be an* acylic trace *if:*

$$\forall q_i, q_j, i < j < m, q_i \neq q_j$$

Given two traces, we define the following notion of prefix of a trace.

DEFINITION 1.12.– *Given a trace* $T = q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$*, the* prefix of length n *of* T *is the trace denoted by* $|T|_n$ *and defined as follows:*

$$|T|_n = \begin{cases} q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{n-1}}{\Rightarrow} q_n & \text{if } n < m \\ q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m & \text{otherwise} \end{cases}$$

Similarly, we say that a trace T_1 *is a* prefix *of a trace* T_2 *if there exists* $n \geq 0$ *such that* $|T_2|_n = T_1$*.*

We now define the following notion of trace set.

DEFINITION 1.13.– *Given a timed automaton* $\mathcal{A}$*, the* trace set of $\mathcal{A}$ *refers to the set of traces associated with the runs of* $\mathcal{A}$*.*

Often, when depicting trace sets, we will not depict each trace separately, but depict the trace set under the form of a tree or a graph. Note, however, that this graph structure is only used for the sake of simplicity of representation of the possible traces, and does not contain any information on the possible *branching* behavior of the system.

EXAMPLE 1.4.– The trace set associated with the timed automaton of example 1.1 is shown in Figure 1.4.

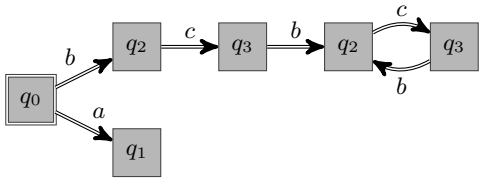


Figure 1.4. *Example of a trace set of a timed automaton*

This trace set contains an infinite number of finite traces. Note also that it is obviously not acyclic, because there are (actually infinitely many) traces passing several times by locations q_2 and q_3.

We extend the notion of acyclicity of a trace to trace sets, and say that a trace set is *acyclic* if all its traces are acyclic. We also say that a location q belongs to the trace set of $\mathcal{A}$ if it belongs to a trace of the trace set of $\mathcal{A}$.

In the following, we are interested in verifying properties on the trace set of $\mathcal{A}$. For example, given a predefined set of "bad locations", a reachability property is satisfied by a trace if this trace never contains a bad location; such a trace is "good" with respect to this reachability property. A trace can also be said to be "good" if a given action always occurs before another one within the trace (see example in section 2.1.1). Actually, the good behaviors that can be captured with trace sets are relevant to *linear-time properties* [BAI 08], which can express properties more general than reachability properties.

DEFINITION 1.14.– *Given a timed automaton $\mathcal{A}$, and a property on traces, we say that a trace of $\mathcal{A}$ is* good *if it satisfies the property, otherwise it is* bad*. Likewise, we say that the trace set of $\mathcal{A}$ is* good *if all its traces are good, otherwise it is bad.*

1.4. Parametric timed automata

Parametric timed automata are an extension of the class of timed automata to the parametric case. Parametric timed automata allow within guards and invariants the use of parameters in place of constants [ALU 93c]. This model is interesting when we do not only want to check that a system is correct for *one* value of the constants, but for a whole dense set of values. The model of parametric timed automata is also interesting to synthesize parameters for which a given property is satisfied.

Unfortunately, for most interesting problems, parametric timed automata lose the decidability results proved for timed automata. In particular, the reachability of a state is not decidable (although semi-algorithms do exist, i.e. if the algorithm terminates, then the result is correct). Moreover, parametric timed automata are even more sensitive to the state space explosion problem because of the addition of the parameters. In practice, this comes also from the fact that the data structures used to represent parametric timed automata are far less efficient than the data structures used for timed automata (typically difference bound matrices, proposed in [BER 83] for the analysis of time Petri nets, and introduced in [DIL 89] for timed automata). Structures handling parametric timed models include parametric difference bound matrices (an extension of difference bound matrices, proposed in [HUN 02]), SAT-solvers, SMT-solvers and polyhedra. Avoiding the explosion of the state space, and finding cases for which analyses are decidable for parametric timed automata, are actually some of the motivations for the techniques described in this book.

1.4.1. *Syntax*

DEFINITION 1.15.– *A* parametric timed automaton $\mathcal{A}$ *is a* 8*-tuple of the form* $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$*, where:*

- Σ *is a finite set of* actions*,*
- Q *is a finite set of* locations*,*
- $q_0 \in Q$ *is the* initial location*,*
- X *is a set of* clocks*,*
- P *is a set of* parameters*,*
- K *is an* initial constraint *on the parameters of* P*,*
- $I : Q \rightarrow \mathcal{L}(X \cup P)$ *is the* invariant*, assigning to every* $q \in Q$ *a constraint* $I(q)$ *on the clocks and the parameters and*
- $\rightarrow$ *is a transition relation consisting of elements of the form* (q, g, a, ρ, q')*, also denoted by* $q \overset{g,a,\rho}{\rightarrow} q'$*, where* $q, q' \in Q$*,* $a \in \Sigma$*,* $\rho \subseteq X$ *is a set of clock variables to be* reset *by the transition, and* $g \in \mathcal{L}(X \cup P)$ *is the transition* guard*.*

The initial constraint K is useful to define constrained models, where some parameters are already related. For example, in a timed model with two parameters min and max, we may want to constrain min to be always smaller or equal to max, that is $K = \{min \leq max\}$. Although it does not add expressive power (this constraint could be "simulated" by simply adding it to the invariant of the initial location), it is largely used in practice. All case studies considered here make use of this initial constraint. Furthermore, this constraint K can be refined in order to constrain the model further. In the following, given a parametric timed automaton $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$, we will often denote this parametric timed automaton by $\mathcal{A}(K)$ when clear from the context, in order to emphasize that only K will change when performing repeated analysis on refined models of $\mathcal{A}$.

We make use for parametric timed automata of the same graphical representation as for timed automata, that is an oriented graph where vertices correspond to the locations, and edges correspond to the actions. The graphical representation of a parametric timed automaton will be referred to as its *associated graph*.

EXAMPLE 1.5.– We give in Figure 1.5 an example of a parametric timed automaton containing four locations (viz. q_0, q_1, q_2 and q_3), three actions (viz. a, b and c), two clocks (viz. x_1 and x_2) and two parameters (viz. p_1 and p_2). The initial location is q_0.

In this parametric timed automaton, q_0 has invariant $x_1 \leq 5p_1$, q_1 has invariant `true` and both q_2 and q_3 have invariant $x_2 \leq 5p_2$. The transition from q_0 to q_1 has

guard $x_1 \geq 4p_1$ through action a; no clock is reset. The transition from q_0 to q_2 has guard $x_1 \geq 2p_1 \wedge x_2 \geq 3p_2$ through action b and resets clock x_2. The transitions between q_2 and q_3 can be explained similarly.

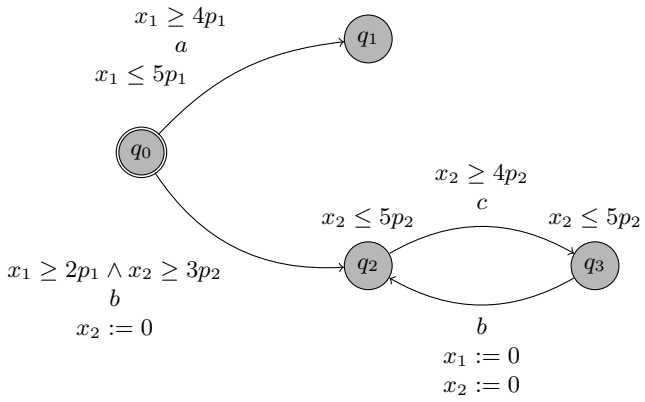


Figure 1.5. *An example of a parametric timed automaton*

1.4.1.1. *Instantiation of a parametric timed automaton*

Given a parametric timed automaton $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ and a parameter valuation $\pi = (\pi(p_1), \ldots, \pi(p_M))$, $\mathcal{A}[\pi]$ denotes the parametric timed automaton $\mathcal{A}(K^\pi)$, where K^π is $K \wedge \bigwedge_{i=1}^{M} p_i = \pi(p_i)$. This corresponds to the parametric timed automaton obtained from $\mathcal{A}$ by substituting every occurrence of a parameter p_i by constant $\pi(p_i)$ in the guards and invariants. We say that p_i is *instantiated* with $\pi(p_i)$. Note that, as all parameters are instantiated, $\mathcal{A}[\pi]$ is a standard timed automaton.

EXAMPLE 1.6.– Consider again the parametric timed automaton $\mathcal{A}$ described in example 1.5. Also consider the following reference valuation π of the parameters: $\pi : \{p_1 = 1, p_2 = 1\}$. Then, the (non-parametric) timed automaton $\mathcal{A}[\pi]$ is the one described in example 1.1.

We now define the notion of *acyclic* parametric timed automaton. This acyclicity of a parametric timed automaton can be deduced purely syntactically from its graphical representation, that is if this representation is acyclic.

DEFINITION 1.16.– *We say that a parametric timed automaton is* graphically acyclic *(or, more simply,* acyclic*) if its associated graph is acyclic.*

Note that the trace set associated with an acyclic parametric timed automaton is necessarily acyclic. (However, note that if a trace set is acyclic, its parametric timed automaton is not necessarily acyclic.)

1.4.1.2. *Parallel composition of parametric timed automata*

We now introduce the notion of network of parametric timed automata, and show in the following definition how N parametric timed automata can be composed into a single parametric timed automaton, by performing a product of the N automata.

DEFINITION 1.17.– *Let $N \in \mathbb{N}$. For all $1 \leq i \leq N$, let $\mathcal{A}_i = (\Sigma_i, Q_i, (q_0)_i, X_i, P_i, K_i, I_i, \rightarrow_i)$ be a parametric timed automaton. The sets Q_i are mutually disjoint.* A network of parametric timed automata *is $\mathcal{A} = \mathcal{A}_1 \| \ldots \| \mathcal{A}_N$, where $\|$ is the operator for parallel composition defined in the following way. This network of parametric timed automata corresponds to the parametric timed automaton $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$, where:*

- $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$,
- $Q = \Pi_{i=1}^{N} Q_i$,
- $q_0 = \langle (q_0)_1, \ldots, (q_0)_N \rangle$,
- $X = \bigcup_{i=1}^{N} X_i$,
- $P = \bigcup_{i=1}^{N} P_i$,
- $K = \bigwedge_{i=1}^{N} K_i$,
- $I(\langle q_1, \ldots, q_N \rangle) = \bigwedge_{i=1}^{N} I_i(q_i)$ *for all* $\langle q_1, \ldots, q_N \rangle \in Q$,

and $\rightarrow$ is defined as follows. For all $a \in \Sigma$, let T_a be the subset of indices $i \in 1, \ldots, N$ such that $a \in \Sigma_i$. For all $a \in \Sigma$, for all $\langle q_1, \ldots, q_N \rangle \in Q$, for all $\langle q'_1, \ldots, q'_N \rangle \in Q$, $(\langle q_1, \ldots, q_N \rangle, g, a, \rho, \langle q'_1, \ldots, q'_N \rangle) \in \rightarrow$ if:

- *for all $i \in T_a$, there exist g_i, ρ_i such that $(q_i, g_i, a, \rho_i, q'_i) \in \rightarrow_i$, $g = \bigwedge_{i \in T_a} g_i$, $\rho = \bigcup_{i \in T_a} \rho_i$, and,*
- *for all $i \notin T_a$, $q'_i = q_i$.*

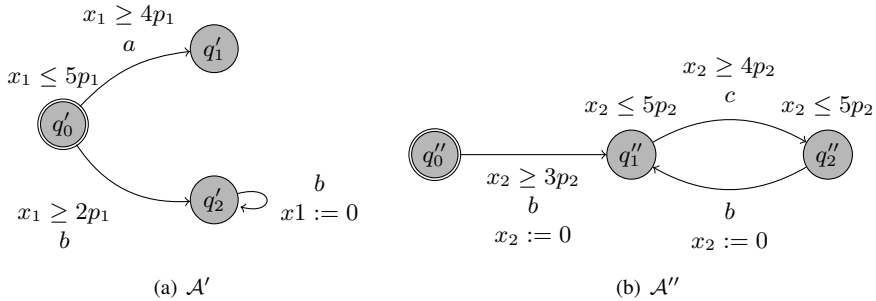
In this definition, the set of actions is the union of the "local" sets of actions (i.e. of each automaton $\mathcal{A}_i$), and similarly for the sets of clocks and parameters. The set of locations of the resulting automaton is the product of the local sets of locations: hence, each location of the resulting automaton is composed of a location of each local automaton. The initial location is made of the initial location of each local automaton. The initial constraint is the intersection of the local initial constraints. Each invariant is the intersection of the local invariants associated with a local location. Finally, a global transition can be taken as follows. For an action a, we compute the set of local automata (denoted by T_a) such that $a \in \Sigma_i$. Then, this transition can be taken if there exists a successor location through a for each automaton in T_a. The guard is obtained by intersecting the local guards of T_a, and the set of clocks to reset is the union of

the local sets of clocks to reset. The local location of automata not in T_a remains unchanged.

Sometimes we meet in practice the requirement that the set of clocks and parameters of each of the timed automata in parallel must be mutually disjoint. Here, for the sake of generality, we do allow the shared use of clocks and parameters between different automata.

Note that, in practice, most tools perform an *on-the-fly* (and partial) composition of the product. Indeed, the computation of this product is usually prohibitively expensive. For example, the composition of 10 automata with 10 locations will each result in a global automaton with a set of locations of size 10^{10}. Hence, most tools only compose the state space that is effectively reached, by computing each state when needed only.

EXAMPLE 1.7.– We give in Figure 1.6 an example of a network of two parametric timed automata.



(a) $\mathcal{A}'$ (b) $\mathcal{A}''$

Figure 1.6. *Example of a network of parametric timed automata*

The composition of those two parametric timed automata in parallel (viz. $\mathcal{A}' \| \mathcal{A}''$) corresponds to the parametric timed automaton from example 1.5, where $q_0 = (q'_0, q''_0)$, $q_1 = (q'_1, q''_0)$, $q_2 = (q'_2, q''_1)$ and $q_3 = (q'_2, q''_2)$.

1.4.2. *Semantics*

We now define the semantics of parametric timed automata. We first introduce the notion of symbolic state.

DEFINITION 1.18.– *Let $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ be a parametric timed automaton. A* (symbolic) state *s of $\mathcal{A}(K)$ is a pair (q, C), where $q \in Q$, is a location, and $C \in \mathcal{L}(X \cup P)$ a constraint on the clocks and the parameters.*

For each valuation π of the parameters P, we may view a symbolic state $s = (q, C)$ as the set of pairs (q, w) where w is a clock valuation such that $<w, \pi> \models C$.

We define the inclusion of a state in another one as the equality of locations and inclusion of constraints.

DEFINITION 1.19.– *We say that a state $s_1 = (q_1, C_1)$ is* included *in a state $s_2 = (q_2, C_2)$, denoted by $s_1 \subseteq s_2$, if $q_1 = q_2$ and $C_1 \subseteq C_2$.*

We say that two states $s_1 = (q_1, C_1)$ and $s_2 = (q_2, C_2)$ are equal*, denoted by $s_1 = s_2$, if $q_1 = q_2$ and $C_1 = C_2$.*

We now define the inclusion of a set S_1 of states in another set S_2. Observe that this notion does not refer to the inclusion of each state of S_1 into a state of S_2, but to the *equality* of each state S_1 with a state S_2. Hence, all states of S_1 exactly appear in S_2.

DEFINITION 1.20.– *We say that a set of states S_1 is* included *into a set of states S_2, denoted by $S_1 \sqsubseteq S_2$, if*

$\forall s : s \in S_1 \Rightarrow s \in S_2$.

We say that two sets of states S_1 and S_2 are equal*, denoted by $S_1 = S_2$, if $S_1 \sqsubseteq S_2$ and $S_2 \sqsubseteq S_1$.*

In this book, we will be interested in checking whether the constraint associated with a symbolic state is satisfied by a given valuation of the parameters. This refers to the following notion of π-compatibility.

DEFINITION 1.21.– *Let $\mathcal{A}$ be a parametric timed automaton, and $s = (q, C)$ be a state of $\mathcal{A}$. The state s is said to be* compatible with π *(or, more simply,* π-compatible*) if $\pi \models C$, and* π-incompatible *otherwise.*

The *initial state* of $\mathcal{A}(K)$ is a symbolic state s_0 of the form (q_0, C_0), where $C_0 = K \wedge I(q_0) \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$. (Recall that H is the number of clocks.) In this expression, K is the initial constraint on the parameters, $I(q_0)$ is the invariant of the initial state, and the rest of the expression lets clocks evolve from the same initial value.

The semantics of parametric timed automata is given in the following under the form of a labeled transition system.

DEFINITION 1.22.– *Let $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ be a parametric timed automaton. The* symbolic semantics of $\mathcal{A}$ *is the labeled transition system $(S, S_0, \Rightarrow)$ over Σ where*

$$S = \{(q, C) \in Q \times \mathcal{L}(X \cup P) \mid C \subseteq I(q)\},$$
$$S_0 = \{(q_0, K \wedge I(q_0) \wedge \textstyle\bigwedge_{i=1}^{H-1} x_i = x_{i+1})\}$$

and a transition $(q, C) \stackrel{a}{\Rightarrow} (q', C')$ *belongs to* $\Rightarrow$ *if* $\exists C'' : (q, C) \stackrel{a}{\rightarrow} (q', C'') \stackrel{d}{\rightarrow} (q', C')$*, with:*

- *discrete transitions* $(q, C) \stackrel{a}{\rightarrow} (q', C')$ *if there exists* $(q, g, a, \rho, q') \in \rightarrow$ *and*

$$C' = \Big(\big(C(X) \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(q')(X')\Big)_{[X' \leftarrow X]}, \text{ and}$$

- *delay transitions* $(q, C) \stackrel{d}{\rightarrow} (q, C')$ *with* $C' = C\uparrow \wedge I(q)(X)$.

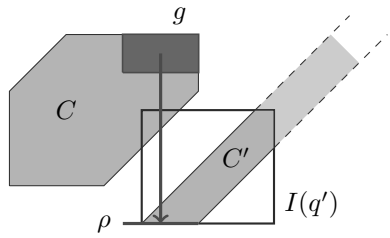


Figure 1.7. *Forward reachability for timed automata*

Let us explain this definition. A transition in the symbolic semantics is the combination of a discrete transition followed by a delay transition. A discrete transition can be taken as follows: first, the original constraint C is intersected with the guard g. We use notation $g(X)$ to denote that the set of variables is X; similarly, $I(q)(X')$ is used to denote that the set of variables is X'. Then, the reset operation is performed by $X' = \rho(X)$; we use $X' = \rho(X)$ to denote that all clocks of X' are equal to clocks of X, except clocks belonging to the set ρ, that are equal to 0. Then, variables in X are eliminated (using, e.g. Fourier–Motzkin elimination [SCH 86]), which is denoted by the projection onto $X' \cup P$. The constraint is then intersected with the invariant of the destination location $I(q')$. Finally, clocks in X' are renamed with X, to get a constraint C' on X and P. A delay transition is obtained by delaying the constraint, using the time elapsing operation, and intersecting it with the invariant of the destination location $I(q)$.

In Figure 1.7, we present in a graphical way the computation of the successor constraint of a state (q, C). First, C is intersected with the guard g of the transition. Then, the clocks that must be reset by the transition (as in ρ) are projected onto zero. Then, the constraint is intersected with the invariant of the destination location $I(q')$. Time elapsing is then applied. The resulting constraint C' is finally obtained by intersecting again with the invariant of the destination location $I(q')$.

DEFINITION 1.23.– *A step of the semantics of a parametric timed automaton $\mathcal{A}(K)$ will be referred to as a* symbolic step *of $\mathcal{A}(K)$. Similarly, a run of the semantics of a parametric timed automaton $\mathcal{A}(K)$ will be referred to as a* symbolic run *of $\mathcal{A}(K)$.*

A symbolic run of $\mathcal{A}(K)$ is a finite alternating sequence of symbolic states and actions of the form $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$ such that for all $i = 0, \ldots, m-1$, and $a_i \in \Sigma$ we have that $s_i \overset{a_i}{\Rightarrow} s_{i+1}$ is a symbolic step of $\mathcal{A}(K)$.

A symbolic run is represented under the form of a branch where states are shown within nodes containing the name of the location and the constraint on the clocks and the parameters, and transitions are shown using edges labeled with the name of the action.

EXAMPLE 1.8.– Consider again the parametric timed automaton $\mathcal{A}$ of example 1.5. Then, Figure 1.8 shows an example of a symbolic run of $\mathcal{A}$.

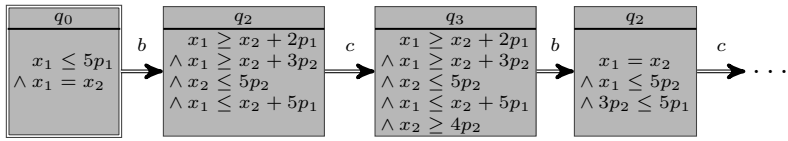


Figure 1.8. *Example of a symbolic run for a parametric timed automaton*

We give below some results on the constraints on the parameters associated with the symbolic runs of a parametric timed automaton, which will be used later on.

We first show that the constraint on the *parameters* associated with the symbolic states becomes more restrictive within a run, i.e. the constraint on the parameters of a given state of a run is included into the constraint on the parameters of a previous state of the same run. The parameter constraints associated with the reachable states can only get stronger, since the parameters do not evolve under the time elapse operation, and can only be further constrained by invariants or guard conditions.

LEMMA 1.1.– *Let $\mathcal{A}(K)$ be a parametric timed automaton and R be a symbolic run of $\mathcal{A}$ of the form $(q_0, C_0) \overset{a_0}{\Rightarrow} \cdots (q_i, C_i) \overset{a_i}{\Rightarrow} (q_{i+1}, C_{i+1}) \overset{a_{i+1}}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$. Then $C_{i+1}{\downarrow_P} \subseteq C_i{\downarrow_P}$, for all $0 \leq i \leq m-1$.*

PROOF. From the semantics of parametric timed automata, C_{i+1} can be computed from C_i by addition of new constraints on the clocks and the parameters and elimination of clock variables only. Thus, C_{i+1} is more restrictive.

Note that the above result does not mean that the constraints on the *clocks and the parameters* become more restrictive within a run due to the elimination of clock variables. The relation $C_{i+1} \subseteq C_i$ does not hold in the general case.

We now state that, given a parametric timed automaton $\mathcal{A}(K)$, the constraint on the *parameters* associated with any symbolic state of $\mathcal{A}$ is included into K.

LEMMA 1.2.– *Let $\mathcal{A}(K)$ be a parametric timed automaton, and (q, C), a symbolic state of a symbolic run of A. Then $C{\downarrow_P} \subseteq K$.*

PROOF. From the definition of the initial state (q_0, C_0) (see definition 1.22), we have $C_0{\downarrow_P} \subseteq K$. The result is then obtained by induction on lemma 1.1.

These two lemmas are the basis for the inverse method, which is described in Chapter 2.

1.4.2.1. *Reachability and post-computation*

Recall from definition 1.7 that a symbolic state s is reachable in one step from another symbolic state s' if s is the successor of s' in a symbolic run. This definition extends to sets of states. One defines $Post^i_{\mathcal{A}(K)}(S)$ as the set of states reachable from a set S of states in *exactly* i steps, and $Post^*_{\mathcal{A}(K)}(S)$ as the set of all states reachable from S in $\mathcal{A}(K)$ (i.e. $Post^*_{\mathcal{A}(K)}(S) = \bigcup_{i\geq 0} Post^i_{\mathcal{A}(K)}(S)$).

In this book, we will be, in particular, interested in computing the set $Post^*_{\mathcal{A}(K)}(\{s_0\})$, where s_0 is the initial state of $\mathcal{A}(K)$. Note that if $Post^{i+1}_{\mathcal{A}(K)}(\{s_0\}) \sqsubseteq \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\}))$, then $Post^*_{\mathcal{A}(K)}(\{s_0\}) = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$.

1.4.2.2. *Traces*

The notion of trace associated with a concrete run, and the notion of trace set associated with a timed automaton apply in a straightforward manner to parametric timed automata.

DEFINITION 1.24.– *Given a parametric timed automaton $\mathcal{A}$ and a symbolic run r of $\mathcal{A}$ of the form $(q_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$, the* trace associated with *r is the alternating sequence of locations and actions $q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$. We say that location q_i, for $1 \leq i \leq m$, belongs to the trace.*

Note that the traces associated with symbolic runs of parametric timed automata are the same mathematical object (i.e. alternating sequences of locations and actions) as the traces associated with concrete runs of timed automata. As a result, we extend the notions of acyclicity and prefixes, defined for traces associated with concrete runs, to traces associated with symbolic runs. Moreover, we show them under the same graphical form as traces associated with concrete runs, that is boxed nodes labeled with locations and double arrows labeled with actions.

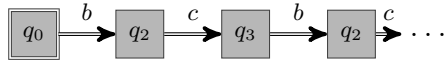


Figure 1.9. *Example of a trace associated with a symbolic run of a parametric timed automaton*

EXAMPLE 1.9.– The trace associated with the symbolic run of example 1.8 is shown in Figure 1.9.

DEFINITION 1.25.– *We say that two (symbolic or concrete) runs are* equivalent*, if their associated trace is equal.*

DEFINITION 1.26.– *Given a parametric timed automaton $\mathcal{A}$, the* trace set of $\mathcal{A}$ *refers to the set of traces associated with the runs of $\mathcal{A}$.*

As for the traces associated with concrete runs, we extend the notion of acyclicity of a trace to trace sets and say that a trace set is *acyclic* if all its traces are acyclic.

EXAMPLE 1.10.– The trace set associated with the parametric timed automaton of example 1.5 is shown in Figure 1.10.

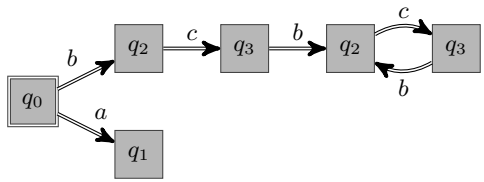


Figure 1.10. *Example of a trace set of a parametric timed automaton*

1.5. Related work

We discuss in this section several approaches to model distributed timed systems. We first justify our choice for the dense-time formalism in section 1.5.1. We then recall timed automata in section 1.5.2 and show some of the advantages of such a formalism. We present time Petri nets and compare them to timed automata in section 1.5.3. We also recall hybrid systems in section 1.5.4 and compare them to timed automata.

1.5.1. *Representation of time*

When modeling timed systems, two major representations of time are used in the literature: the discrete time representation and the dense (or continuous) time representation. In the discrete time model, events can happen only at the integer time

values. This allows the designer to describe the behavior of synchronous systems, where all components are driven by a single common global clock. This discrete time representation is the traditional model for synchronous hardware verification, where events (i.e. changes of signals) happen only at clock ticks, that is at the integer time values. On the contrary, in the dense time representation, events can occur at any real (or rational) time value. As a result, formalisms making use of a dense time representation are usually more complex than formalisms using a discrete representation, but also more expressive.

In this book, we focus on the dense time representation, and more specifically on the formalism of timed automata. A classical formal justification for the dense time model can be found in [ALU 92a]. Furthermore, in this book, we will be in particular interested in verifying asynchronous circuits, where events (changes of signals) can happen at any real-time value. As a result, the dense time representation is certainly more suitable than the discrete time representation. However, one could argue that, for a discrete time with enough precision (i.e. using time steps small enough), the discrete time representation can be suitable for the study of asynchronous circuits. Actually, it was shown that, in certain cases, finding the appropriate time step can be as difficult as doing the real-time model checking. The main interest of dense time representation, though, is that it gives a criterion of *robustness*. The discrete time representation can prove the correctness of a timed system for several integer values of the timing delays, but no information can be given in between two integers, whereas the dense time representation is able to guarantee *intervals* of values for which the system is correct. This is of particular interest in the sense that, when implementing a timed system, timing delays may slightly differ from the (exact and punctual) integer value they have been designed for.

1.5.2. *Timed automata*

Timed automata [ALU 94] have been used to model and successfully verify various case studies, e.g. communication protocols [DAR 97, COL 01], and allowed famous bug discoveries (e.g. [HAV 97]). It has been shown that model-checking properties expressed using the Timed CTL logic [ALU 93a] are decidable [ALU 93a, HEN 94] for timed automata and some of their extensions (e.g. [BOU 04]).

Recent work on timed automata focused on the notion of *robustness*, or implementability. As mentioned in [DE 04], timed automata consider perfect clocks with infinite precision while implementations can only access time through finitely precise clocks. Moreover, timed automata react instantaneously to events while implementations can only react within a given usually small, but not zero, reaction delay. Also note that timed automata may describe control strategies that are unrealistic, such as zeno-strategies or strategies that ask the controller to act faster

and faster [CAS 02]. As a result, models that have been proven correct may not be implementable. A first notion of robust timed automata has been considered in [GUP 97], with the following semantics: "if a robust timed automaton accepts a trajectory, then it must accept neighboring trajectories also and if a robust timed automaton rejects a trajectory, then it must reject neighboring trajectories also". The authors show in particular that the emptiness problem for robust timed automata is still decidable. Another work introducing the notion of "implementable" timed automata is given in [DE 04]. This work is based on the "Almost ASAP semantics" defined in [DE 05]. This semantics relaxes the classical semantics of timed automata in the sense that it does not impose a transition to be taken instantaneously but within a (very small) amount of time. The authors also show in [DE 04] that the notion of robustness defined in [PUR 00] is closely related to their notion of implementability. Robust model checking in this framework has been considered in [BOU 06] with results of decidability. Any reader interested in robustness questions can refer to [MAR 11] for a survey.

The parametrization of timed automata into parametric timed automata [ALU 93c], where parameters are used in guards and invariants in place of constants, allows parametric model checking. In other words, instead of checking if a given location can be reached, or if a given formula expressed, using for example the timed computation tree logic (TCTL) is satisfiable, we can synthesize sets of values for the parameters under which the location can be reached (or under which the formula is satisfied). Unfortunately, most interesting problems related to parametric timed automata have been shown to be undecidable for non-trivial parametric timed automata; this is in particular the case of the emptiness problem [ALU 93c]. We mention work related to the synthesis of parameters in the framework of parametric timed automata in section 2.4.5 and perform a survey of model checkers for several classes of timed automata and their extensions in section 3.10.

1.5.3. *Time Petri nets*

Time Petri nets [MER 74] are a classical and widely used extension of Petri nets for modeling timed distributed systems using places, tokens and transitions that can be fired within a time interval. It has been shown that state reachability is decidable for bounded time Petri nets. Both time Petri nets and timed automata are dense-time formalisms, which allow to study and verify dense-timed models, for example asynchronous circuits. As a result their underlying state space is infinite and verification techniques that enumerate exhaustively the state space cannot be applied. The main difference relies in the fact that, although both formalisms may be considered as infinite state because of the real-valued time values, the number of locations in timed automata is *finite*, whereas time Petri nets remain an infinite marking model. Although the number of places of a time Petri net is bounded, the number of tokens in each node is (in the general case) unbounded, thus leading to a

potentially infinite number of markings. Note, however, that several subclasses of Petri nets consider a bounded number of tokens per place. Several classes of time Petri nets were shown to be equivalent to several classes of timed automata, and several approaches for translations from time Petri nets to timed automata have been proposed (see a survey in [PEN 06] as well as more recent work, e.g. [CAS 06, DAP 07, LIM 09]).

A parametrization of time Petri nets with stopwatches (i.e. an extension of traditional clocks that can be suspended and resumed) has been considered in [TRA 09]. Similarly to parametric timed automata, the parameters are used instead of constants in the firing conditions associated with the transitions. The authors propose semi-algorithms for the synthesis of parameter valuations satisfying a formula expressed using a subset of parametric TCTL formulas.

1.5.4. *Hybrid systems*

Hybrid systems can be seen as a generalization of timed automata, where "clocks" (actually, real-valued variables) may evolve at different rates. Those variables do not necessarily model the time elapsing, but can represent any real-valued continuous variable, such as temperature, speed and geographical position. Hybrid automata were introduced in [ALU 93b] and allow us to define in any location a law of evolution with respect to time for each of the dynamic variables. A common subclass of hybrid systems consists of *linear hybrid systems*, where the derivatives of the variables are given within a (constant) interval for each location. An interesting survey on the different models of hybrid systems and linear hybrid automata, mostly in the 1990s, is given in Chapter 4 of [FRE 05]. Furthermore, work related to the parameter synthesis for hybrid systems is discussed in section 5.5.