Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

Specification and Verification using Timed Automata

Luca Tesei

Supervisor
Roberto Barbuti

Abstract

We present some original results in the framework of the specification and of the verification of real-time systems. The underlying formal model of computation is the timed automata model. It is formally introduced in detail and within a theory of timed languages. Different aspects and variants of timed automata are exposed. The main verification techniques that have been proposed in the literature are shown in detail. Particular attention is dedicated to the simplifications that have been done to the theoretical model of timed automata in order to construct simulators and/or model checking tools. On the side of specification, the thesis presents two extensions of the original model of timed automata: non-instantaneous actions and urgent transitions. The first extension improves the expressive power of the formalism of timed automata and allow more precise specifications. The second one is a very useful tool in the task of the specification of typical behaviors of real-time systems. It allows writing clear and concise specifications. We show that its introduction does not increase the expressive power of timed automata. Both the extensions are defined within the theoretical model of timed automata, but it is shown that they have a similar definition also for the implementable timed automata. Moreover, they are used to specify a classical railway cross example and a multicast protocol for mobile computing. On these example we perform automatic verifications using existing model-checking tools. On the side of verification, we define a timed version of a classical security property defined on untimed systems: the non-interference property. We show how this formulation can be used to state and verify time-dependent security properties of timed systems. The definition of the timed non-interference property is given in two different formulations. The first is the clearer and more natural one, but suffers of a negative result of undecidability. The second one is effectively used in an analysis of the strength of a time-dependent mutual exclusion protocol against timed attacks.

Acknowledgments

First of all I want to thank my supervisor Roberto Barbuti for his support and help started with my graduation thesis and continued along my Ph.D. course. I am grateful to Nicoletta De Francesco and Antonella Santone for their collaborations and suggestions in the last years.

I thank my external referees Danièle Beauquier and Manfred Droste for their careful reading of my thesis, their comments and their suggestions of corrections. I also thank the anonymous referees of the published and unpublished papers on which a part of this thesis is based, for their criticisms and suggestions.

Let me switch to Italian... per ringraziare i miei genitori, Mario ed Olivetta, e la mia famiglia per il loro supporto e sostegno continui.

Un grazie particolare va agli amici con cui ho condiviso la mia vita qui a Pisa in questi anni di università e di dottorato: a Simone, Enzo e Domenico per la loro amicizia e per avermi sempre stimolato nelle relazioni sociali, a Diego e Gianluca per aver condiviso con me molti momenti di crescita, ad Alessandro per i discorsi speciali che abbiamo fatto e continuiamo a fare, a Francesco per avermi sempre sopportato anche nei periodi bui, a Sauro per la sua amicizia nei periodi passati a Macerata, a tutti gli altri amici ed amiche con cui ci siamo divertiti ed aiutati.

Contents

List of Figures

Introduction

Formal methods for the specification and the algorithmic verification of systems have been of great impact on industry and on real-life applications. The fundamental idea of these methods is as follows: define mathematical models to represent systems, their behaviors and also the safety, liveness, fairness, structural requirements. Then, using the precise mathematical definitions, construct (possibly efficient) algorithms that accept a model and a requirement and that perform an analysis determining if the model meets its specifications. The correctness of the answer of the algorithms can be established by a mathematical proof. However, this approach could fail if drawbacks are present in the task of modeling, i.e. the model could not be consistent with the actual system.

A class of systems for which this approach has been adopted and studied is the class of *reactive systems*. Examples of reactive systems are air traffic controllers, dishwashers controllers, cash dispensers, operating systems, etc. A reactive system is characterized by the fact that it operates in an environment that communicates with it by sending requests or by signaling events. The system performs its tasks in order to satisfy its requirements when inserted in the environment for which it has been developed. This means that, when describing a reactive system, also environments events have to be modeled. Moreover, a reactive system is supposed to operate forever, i.e. its behaviors are infinite.

A reactive system can be modeled at different levels of abstractions and, usually, its model is a composition of several modules that operate concurrently. Generally, in the models, the computational tasks of the system are abstracted away to focus only on the concurrent and/or communications aspects.

Several formalisms have been proposed in the past to model reactive systems. Consider, for instance, transition systems [Kel76, Pnu77], process algebras [Mil80, Hoa78], I/O automata, Petri nets etc. Also formalisms to express the requirements of the systems have been proposed, the most important being temporal logics, first introduced in [Pnu77]. In the 80's many algorithms for the so-called model-checking problem have been developed successfully. The model-checking consists in determining if a certain mathematical structure representing a system is a model of a formula of a temporal logic representing a requirement. One important aspect of model-checking algorithms is that they give diagnostic information if the verification fails. For this reason many tools have been developed for the model-checking of different logics with respect to different formalisms of specification. They represent a very useful aid in the design of complex systems.

The formalisms recalled above have an important limitation. They allow to specify the behaviors of reactive systems only in a qualitative way. In other words, within these formalisms, that we will call *untimed formalisms*, we cannot specify quantitative time information about the behaviors of the modeled system. For instance, we can specify a system in which "a request a is followed by a response b or by a *fail* event", but we cannot specify timed behaviors as "a request a is followed by a response b within 5 time units or, exactly after 6 time units, by a *fail* event". The reactive systems in which such information on the actual times is a crucial aspect in defining their behaviors and their requirements are called *real-time systems*.

A natural way to apply formal methods of verification to real-time systems is to add, to the existing models, a suitable notion of time. This task started at the end of 80's and, since those years, researchers have proposed several timed versions of the existing untimed formalisms. Examples of this extensions are timed Petri nets [Ram74, GS94], timed I/O automata [LA92], timed transition systems [Ost90, AH92b, HMP94] and timed process algebras [NS94]. Also logical languages have been developed to express quantitative timing properties. For instance, the linear-time logic PTL (Propositional Temporal Logic) [GPSS80] has been extended to its timed version, TPTL, in [AH94]. The branching-time logic CTL (Computational Temporal Logic) [CES86] has been extended to TCTL in [ACD93]. Together with logic languages comes a lot of model checking algorithms and techniques [EMSS90, AH89, ACD94, AFH96, AD90, Lew90, Ost90, AH90, HLP90].

An important aspect of a timed formalism is the time domain that it uses. Many of the formalisms above use a discrete time domain, i.e. the time is represented by integer numbers. This choice simplifies the process of verification because the untimed verification algorithms can be simply adapted to handle such a type of time domain. However, formalisms that use a *dense* time domain can represent in a more realistic way the actual physical processes that real-time systems are supposed to control or to deal with.

In this thesis we adopt, as referring model for the specification of real-time systems, the model of *timed automata*. It was introduced in [AD90, AD94] by Rajeev Alur and David L. Dill. Since their presentation, timed automata have been widely studied from different points of view [ACH$^+$92, ACH94, AFH99, AH92a], in particular for their possible use in the verification of real-time systems [ABBL98, ABG98, ACD93, AH94, HK94, HNSY94, Yov96].

Chapters 1 and 2 of this thesis are dedicated to a detailed introduction of timed automata and the main verification techniques of real-time systems modeled with them. More precisely, we start defining a fundamental structure, that, following [AD94], we call timed transition table, which is the basis of the different notions of automata that we introduce. We address several issues regarding the dense time domain used by timed automata and some aspects of timed behaviors concerning its use, e.g. divergence, are treated.

Then, timed Büchi automata are defined and the theory of timed regular languages is introduced. We show that the concept of timed Büchi automata is a high-level powerful formalism to represent linear-time trace-based timed behaviors of real-time systems represented by timed languages. The parallel composition operator for timed Büchi automata is defined precisely starting from the construction outlined in [AD94] and its properties are formally proved. Several classes of (variants of) timed automata are defined and their expressive power, given in terms of accepted timed languages, is used to relate them.

In Chapter 2 we show the possible verifications that can be performed within the automata-theoretic framework of timed Büchi automata and we discuss the problems that arise in an effective implementation of this model. We define timed safety automata as a simplified formalism, with some limitations with respect to time Büchi automata, that is the referring *implementable* variant of timed automata. We show how the model-checking of the real-time temporal logic TCTL can be performed with respect to timed safety automata. Then, we introduce two existing automatic tools that perform model-checking verification for weak versions of TCTL with respect to slight variants of timed safety automata, namely KRONOS [DOTY96] and UPPAAL [LPY97, BLL+96]. They are used in Chapters 3, 4 and 5 to perform some verifications on the examples.

The model presented in Chapter 1 has enough expressive power to allow the specification of most of the typical behaviors of real-time systems. However, not always the basic model is satisfactory when modeling some features that are indeed often seen in real-time systems. It can happen that some behaviors cannot be expressed in the basic model, or that some characteristics have to be simulated with the available machinery resulting in a complex and hard to understand specification. For these reasons, many extensions to the basic model have been proposed [BPDG98, CG00, DZ98, GHJ97, LMP00]. All these extensions have been discussed with respect to the expressiveness of the original model.

In this thesis we present two contributions to this research field. Chapters 3 and 4 contains two original extensions of the basic model of timed automata. Both the ideas came out in response to specification necessities. The first of them concerns the duration of actions which, in the basic model, are instantaneous. This constraint is relaxed in order to possibly have actions with a duration. The second extension concerns the possibility to have a priority between the possible transitions in a state of a timed automaton. In the basic model all of the transitions have the same priority and can be executed non-deterministically. We introduce the possibility of specifying that some transitions are urgent, i.e. that they have priority with respect to others and that their execution has to be fired within a short time interval from their enabling.

Both the presented extensions are compared with the original model in terms of expressive power, conciseness and possibility to have more precise, readable and elegant specifications. For each extension a translation procedure that maps an automaton with the new features into a basic one has been defined. This allows the use of all verification techniques and tools developed in the framework of timed automata on the specifications that use the extensions.

Moreover, we show that in both cases the extensions added to timed automata can be easily transported to the model of timed safety automata to make the automatic verification effective and feasible. In this way, the extensions are defined upon a clear theoretical basis and implementation details can be considered a separate task to face to.

To give an evidence of the feasibility of this task we apply the introduced extensions to timed safety automata modeling the classical railway cross example, for non-instantaneous actions, and a multicast protocol for mobile computing, for urgent transitions. On these automata we use KRONOS, in the first case, and UPPAAL, in the second case, to perform the automatic verification of some properties.

The contributions presented in this thesis are not restricted to the side of the specification task that we enriched with the proposed extensions. We present a contribution also in the area of verification. In particular, we follow the research direction that tries to bring useful and fruitful concepts defined for untimed formalisms into the framework of real-time systems.

This is the area within Chapter 5 can be placed. The untimed framework that we address is the so-called *information flow analysis* and the idea we attempt to import is that of the security property called non-interference [GM82]. We start by introducing the idea of non-interference as it has been presented in the literature and then we define a timed notion of non-interference for timed automata. This notion could be considered natural for detecting interference caused by the frequency of certain events in timed systems. However, the presented notion cannot be effectively used because its checking requires to solve the timed automata equivalence problem, which is shown to be undecidable. This problem is solved by defining a decidable notion of timed non-interference that is based on the decidable problem of reachability test.

We illustrate the characteristics of our notions of timed non-interference by suitable examples. In particular, a timed non-interference analysis is carried out to check the strength of the Fischer's mutual exclusion protocol [Lam87], which depends on time, against timed attacks. The analysis is done using both timed automata and timed safety automata. In the latter case the tool UPPAAL is used to support the analysis.

Chapter 1

Timed Automata

Abstract

In this chapter we introduce in detail the model of timed automata. We treat many different aspects including fairness and acceptance conditions, divergence of derivation, parallel composition, expressive power, normal forms and non-observable transitions. The model is introduced modularly starting from the concept of timed transition tables toward timed automata with accepting conditions. The concepts and the notation introduced in this chapter are used in all the subsequent chapters.

Throughout the thesis, we introduce some conventions on notation or on terms. These parts have been highlighted by introducing a numbered type of text paragraphs as the following one, which is the first.

Convention 1 *In the following,* $\mathbb{R}$ *(*$\mathbb{Q}$*) is the set of real (rational) numbers,* $\mathbb{R}^{\geq 0}$ *(*$\mathbb{Q}^{\geq 0}$*) the set of non-negative real (rational) numbers and* $\mathbb{R}^{>0}$ *(*$\mathbb{Q}^{>0}$*) is the set of positive real (rational) numbers. The set of natural numbers is denoted by* $\mathbb{N}$ *and* $\mathbb{N}^{>0}$ *is the set of positive natural numbers.*

1.1 Clock variables

The idea of clock variables is central in the framework of timed automata. They have been introduced in [AD90] and they have been one of the most used tool to introduce time constraints in untimed formalisms. A *clock* is a variable that takes values from the set $\mathbb{R}^{\geq 0}$. The clocks measure time as it elapses. All the clocks of a given system advance at the same rate: when increasing, they can be viewed as functions on time whose derivative is equal to 1. Clock variables, or simply clocks, are ranged over by $x, y, z, \ldots$ and we use $\mathcal{X}, \mathcal{X}', \ldots$ to denote sets of clocks.

A *clock valuation* over $\mathcal{X}$ is a function that assigns a non-negative real number to every clock. The set of valuations of $\mathcal{X}$, denoted by $\mathcal{V}_{\mathcal{X}}$, is the set of total functions from $\mathcal{X}$ to $\mathbb{R}^{\geq 0}$. Clock valuations are ranged over by $\nu, \nu', \ldots$. Given $\nu \in \mathcal{V}_{\mathcal{X}}$ and $\delta \in \mathbb{R}^{\geq 0}$, we use $\nu + \delta$ (resp. $\nu - \delta$) to denote the valuation that maps each clock $x \in \mathcal{X}$ into $\nu(x) + \delta$ (resp. $\nu(x) - \delta$). Note that if there exists $x \in \mathcal{X}$ such that $\nu(x) - \delta < 0$, $\nu - \delta$ is not a clock valuation.

Given two disjoint sets of clock variables $\mathcal{X}$ and $\mathcal{X}'$, and two clock valuations $\nu_1 \in \mathcal{V}_{\mathcal{X}}$ and $\nu_2 \in \mathcal{V}_{\mathcal{X}'}$, we denote by $\nu_1 \cup \nu_2$ a clock valuation in $\mathcal{V}_{\mathcal{X} \cup \mathcal{X}'}$ that is such that

$$\nu_1 \cup \nu_2(x) = \begin{cases} \nu_1(x) & \text{if } x \in \mathcal{X} \\ \nu_2(x) & \text{if } x \in \mathcal{X}' \end{cases}$$

Clock variables can be reset during the evolution of the system when certain actions are performed or certain events occur. The reset consists in instantaneously set the value of a clock to 0. Immediately after this operation the clock restarts to measure time at the same rate as the others. The reset is useful to measure the time elapsed since the action/event that reset the clock occurred. Given a set $\mathcal{X}$ of clocks, a *reset* γ is a subset of $\mathcal{X}$. The set of all resets of clocks in $\mathcal{X}$ is denoted by $\Gamma_{\mathcal{X}}$ and reset sets are ranged over by $\gamma, \gamma', \ldots$ Given a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ and a reset γ, with $\nu \backslash \gamma$ we denote the valuation

$$\nu \backslash \gamma(x) = \begin{cases} 0 & \text{if } x \in \gamma \\ \nu(x) & \text{if } x \notin \gamma \end{cases}$$

1.2 Clock constraints

The timed behavior of a system represented by an automaton is expressed using constraints associated to the edges of the automaton. Such constraints depend on the actual values of the clock variables of the system. Their form, and thus their expressive power, is a crucial issue in the definition of timed automata because the decidability of fundamental verification questions strictly depends on the structure of the clock constraints. We will discuss this question in more details in Section 2.2.2.

Given a set $\mathcal{X}$ of clocks, the set $\Psi_{\mathcal{X}}$ of *clock constraints* over $\mathcal{X}$ are defined by the following grammar:

$$\begin{array}{rcl} \psi & ::= & \mathit{true} \\ & & \mid \mathit{false} \\ & & \mid x \;\#\; c \\ & & \mid x - y \;\#\; c \\ & & \mid \psi \wedge \psi \\ & & \mid \psi \vee \psi \\ & & \mid \neg \psi \end{array}$$

where $x \in \mathcal{X}$, $c \in \mathbb{Q}^{\geq 0}$, and $\#$ is a binary operator in $\{<, >, \leq, \geq, =\}$. Note that we allow the constants c that are compared with the values of clocks to be in the set of rational numbers. In Section 1.6 we will show how we can restrict to constants in $\mathbb{N}$.

Clock constraints are evaluated over clock valuations. The satisfaction of a clock constraint $\psi \in \Psi_{\mathcal{X}}$ by a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$, denoted by $\nu \models \psi$, is defined as follows:

$$\nu \models true$$

$$\nu \not\models false$$

$$\nu \models x \# c \ \text{ iff } \ \nu(x) \# c$$

$$\nu \models x - y \# c \ \text{ iff } \ \nu(x) - \nu(y) \# c$$

$$\nu \models \psi_1 \wedge \psi_2 \ \text{ iff } \ \nu \models \psi_1 \text{ and } \nu \models \psi_2$$

$$\nu \models \psi_1 \vee \psi_2 \ \text{ iff } \ \nu \models \psi_1 \text{ or } \nu \models \psi_2$$

$$\nu \models \neg\psi \ \text{ iff } \ \nu \neg \models \psi$$

The previous definition of clock constraint is redundant. In Section 1.6 we will present some normal forms of timed automata that enable us to restrict to a minimal syntax for the constraints.

1.3 Timed Transition Tables

Now we can formally introduce the concept of the timed transition table of a timed automaton. This is almost a timed automaton: some additional fairness conditions on the runs are needed to obtain a timed automaton from a timed transition table.

Definition 1.1 (Timed Transition Table) *A timed transition table T is a tuple $(Q, \Sigma, \mathcal{E}, B, \mathcal{X})$, where: Q is a finite set of states or locations, Σ is a finite alphabet of symbols, $\mathcal{E}$ is a finite set of edges, $B \subseteq Q$ is the set of initial states, $\mathcal{X}$ is a finite set of clocks.*

Each edge $e \in \mathcal{E}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times Q$. If $e = (q, \psi, \gamma, \sigma, q')$ is an edge, q is the source*, q' is the* target*, ψ is the* constraint*, σ is the* label*, γ is the* reset.

A timed transition table can be used to model a reactive system. The locations in Q represent the states and the labels in Σ are the observable events/actions of the system. The edges represent the instantaneous changes that occur in the system in response of an event or as a consequence of an action. Quantitative real-time requirements can be specified for both events and actions using the clock constraints and the clock resets on the edges. The behaviors of the system are the traces of its observable events.

To obtain traces we have to define the dynamics of a timed transition table. A timed transition table T is a finite structure that represents an infinite labeled transition system (l.t.s) whose runs will be used to define the behaviors of the system.

T1	$$\frac{\delta \in \mathbb{R}^{>0}}{(q,\nu) \xrightarrow{\delta} (q,\nu+\delta)}$$
T2	$$\frac{(q,\psi,\gamma,\sigma,q') \in \mathcal{E}, \nu \models \psi}{(q,\nu) \xrightarrow{\sigma} (q',\nu\backslash\gamma)}$$

Figure 1.1: Rules for the transition relation of $\mathcal{S}(T)$

Definition 1.2 *Given a timed transition table $T = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$, its associated labeled transition system is denoted by $\mathcal{S}(T)$ and is such that $\mathcal{S}(T) = (S, \rightarrow)$, where S is a set of states and $\rightarrow$ is the transition relation. The states S are pairs (q, ν), where $q \in Q$ is a state of T, and $\nu \in \mathcal{V}_{\mathcal{X}}$ is a clock valuation. The transition relation $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}^{>0}) \times S$ is defined by the rules of Figure 1.1.*

Convention 2 *In some papers in the literature, the term "timed transition system" is used to refer to l.t.s. of the form of the previous definition. We will adopt this convention being aware, however, that this term is used also to refer to a different class of l.t.s. See, for instance, [Ost90, AH92b, HMP94, ACH94].*

The timed transition system (t.t.s.) $\mathcal{S}(T)$ models the possible behaviors of T in the following sense. An initial state of $\mathcal{S}(T)$ is a state (q, ν) where $q \in B$ is an initial state of T and ν is the valuation which assigns 0 to every clock in $\mathcal{X}$. At any state q, given a valuation ν, T can stay idle or it can perform an action labeling an outgoing edge e. If T stays idle for $\delta \in \mathbb{R}^{>0}$ time units, $\mathcal{S}(T)$ makes a transition, labeled δ, from the current state to a state in which the location of T is the same, but the valuation has been modified according to the elapsed time (rule T1). We will call these transitions *δ-transitions.*

If T moves along an outgoing edge $e = (q, \psi, \gamma, \sigma, q')$, this corresponds to a transition, labeled by σ, of $\mathcal{S}(T)$ from the state (q, ν) to the state $(q', \nu\backslash\gamma)$. This transition is possible only if the current clock valuation respects the constraint ψ of e (rule T2). We will call these transitions *Σ-transitions.*

Definition 1.3 (Time, event, state sequences) *Let $T = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ be a timed transition table and let $r = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \cdots$ be an* infinite *derivation of its associated timed transition system $\mathcal{S}(T)$ where $s_0 = (q_0, \nu_0)$ is an* initial state, *i.e.*

- *$q_0 \in B$ is an initial state of T and*

- $\forall x \in \mathcal{X}.\ \nu_0(x) = 0$

- *The* time sequence $t_0\, t_1\, t_2 \cdots$ *of the times elapsed from state* s_0 *to every state* s_i *in* r *is defined as follows:*

$$t_0 = 0$$

$$t_{i+1} = t_i + \begin{cases} 0 & \mathit{if}\ l_i \in \Sigma \\ l_i & \mathit{otherwise} \end{cases}$$

- *The* label sequence *of* r *is the sequence of the transitions occurred during* r*, including the elapsed times, from the initial state:* $(l_0, t_0)(l_1, t_1) \cdots$

- *The* action sequence *of* r *is the projection of the label sequence of* r *on the pairs* $\{(l_i, t_i) \mid i \geq 0,\ l_i \in \Sigma\}$

- *The* state sequence *of* r *is the projection of the sequence of states* $s_0\, s_1\, s_2 \cdots$ *of* r *on the states* $\{q_i \in Q \mid s_i = (q_i, \nu_i),\ i = 0\ \vee\ (i \geq 1\ \wedge\ l_{i-1} \in \Sigma)\}$

- *The set of* infinitely many repeated states *of* r *is denoted by* $\mathit{inf}(r)$ *and is the set of locations* $q \in Q$ *that occur infinitely many times in the state sequence of* r*.*

Convention 3 *For the sake of readability, we use, throughout the thesis, the term "derivation", applied to a t.t.s. associated to a timed transition table or to a timed automaton, to denote an* infinite *derivation that* starts at an initial state. *When we want to refer to a different kind of derivation, we use a different term which explicitly describes its characteristics.*

Note that, by definition, the time sequence of every derivation r of $\mathcal{S}(T)$ is non-decreasing:

$$\forall i \in \mathbb{N}.\ t_i \in \mathbb{R}^{\geq 0}\ \wedge\ t_i \leq t_{i+1}$$

In some papers there is a more restrictive requirement, i.e. that the time sequence be a strictly increasing sequence of times. This means that two consecutive Σ-transitions are not allowed or, in other words, that the timed transition table cannot perform two or more actions/events at the same instant of time. It has to stay idle for a certain time $\delta > 0$ and then another Σ-transition can be performed. However, the fundamental properties of timed automata holds regardless of the choice in the monotonicity of the time sequence [AD94].

We want to remark here that the derivations r of $\mathcal{S}(T)$ as defined above can have several consecutive δ-transitions that can be considered equivalent to one δ-transition whose delay time δ is the sum of the delay times of the consecutive transitions. This property has been called *time additivity*. It is a postulate on time

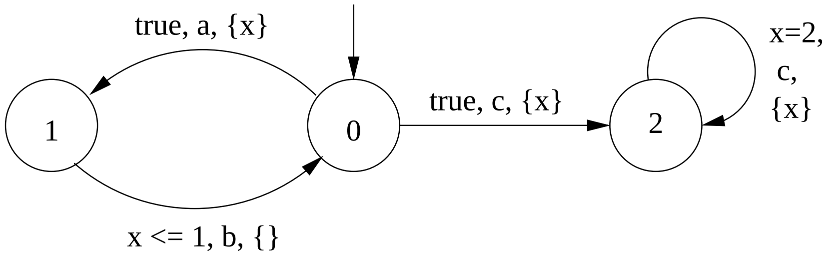

Figure 1.2: A timed transition table T_0

which is assumed, as far as we know, in all proposed models of behaviors of real-time systems.

Let us show a first example of a timed transition table. First note that a timed transition table can be represented as a directed graph in which the nodes are the locations of Q and the directed edges are the edges $\mathcal{E}$. We will often use this representation, as outlined in the following.

Convention 4 *Throughout the thesis we use a graphical representation to specify timed transition tables and timed automata. The graphical notation is the usual one of the classical automata framework enriched with the new notation needed to handle timing operators. Figure 1.2 shows a timed transition table T_0. Circles represent locations which we named 0, 1 and 2. The locations with a dangling ingoing arrow are initial locations. In this case location 0 is the unique initial location. Edges are represented as arrows from the source state to the target state and are labeled with, in order, the clock constraint, the action/event symbol and the reset set. If an arrow is labeled by a set of symbols, it represents a set of edges, one for each symbol in the set, each of which having the same clock constraint and clock reset. In T_0 there is an edge starting from location 0 to location 1 whose clock constraint is true, whose action symbol is a and whose reset set is $\{x\}$. We can deduce from the figure that the only clock variable of T_0 is x and that the alphabet is $\Sigma = \{a, b, c\}$.*

1.3.1 Fair derivations

In T_0 a quantitative time constraint is imposed on the occurrence of b with respect to the occurrence of a. Namely, each a is followed by a b within 1 time unit. This is obtained by resetting the x when a occurs and by adding the constraint $x \leq 1$ in the edge executing b. Note that this is not true for all possible infinite derivations of $\mathcal{S}(T_0)$. As a matter of fact, $\mathcal{S}(T_0)$ could reach a state $(1, \nu)$ such that $\nu(x) > 1$ and then it could start an infinite sequence of δ-transitions yielding an infinite derivation in which only a finite number of Σ-transitions has been performed. The formalism of timed transition tables or timed automata does not consider such derivations as fair derivations.

Definition 1.4 (Fair derivations) *Let $T = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ be a timed transition table and let r be an* infinite *derivation of $\mathcal{S}(T)$.*
Then, r is **fair** *iff $inf(r) \neq \emptyset$.*
Equivalently, r is fair if its action sequence or state sequence is infinite.

Most of the times we will consider as actual behaviors of the system modeled by a timed transition table (or by a timed automaton) *only* those corresponding to fair derivations of its timed transition system. This assumption will not be used in Section 2.3.1 where the model of Timed Safety Automata is introduced. In this model a similar notion of fairness is assured by invariant conditions on the states.

As an example, in every fair derivation of $\mathcal{S}(T_0)$, if T_0 is in state 1 then it cannot let elapse a time greater than 1 time unit, because if this happens, it cannot execute a Σ-transition anymore. Thus, if we consider as behaviors of T_0 only those that derive from fair derivations, then T_0 always executes the edge labeled with b within one time unit from the occurrence of a. A different behavior is not considered a behavior of the system.

However, when modeling real-time systems there could be some cases in which we want to specify a behavior that is unfair. For instance, the specification of a system could say that in certain situations a state q_{div} can be reached in which it is possible to let the time elapse forever. This behavior can be specified with fair derivations simply adding to Σ a new symbol *idle* representing an idle transition and adding to the timed transition table representing the system the edge $(q_{div}, true, \emptyset, idle, q_{div})$.

1.3.2 Specification of sequences

A possible fair derivation of the timed transition system $\mathcal{S}(T_0)$ is the following, where the clock valuation is specified by the value of the unique clock x.
$r_0 = (0, x = 0) \xrightarrow{7.6} (0, x = 7.6) \xrightarrow{1.4} (0, x = 9.0) \xrightarrow{a} (1, x = 0) \xrightarrow{0.5} (1, x = 0.5) \xrightarrow{b}$
$(0, x = 0.5) \xrightarrow{100.55} (0, x = 101.05) \xrightarrow{c} (2, x = 0) \xrightarrow{2} (2, x = 2) \xrightarrow{c} (2, x = 0) \cdots$
The time sequence of r_0 is $t_0 = 0, t_1 = 7.6, t_2 = 9, t_3 = 9, t_4 = 9.5, t_5 = 9.5, t_6 = 110.05, t_7 = 110.05, t_8 = 112.05, t_9 = 112.05 \cdots$
The label sequence of r_0 is $(7.6, 7.6)(1.4, 9)(a, 9)(0.5, 9.5)(b, 9.5)(100.55, 110.05)$ $(c, 110.05)(2, 112.05)(c, 112.05) \cdots$
The action sequence of r_0 is $(a, 9)(b, 9.5)(c, 110.05)(c, 112.05) \cdots$
The state sequence of r_0 is $0, 1, 0, 2, 2, \cdots$

Convention 5 *We can, and almost always will do so, characterize the set of infinite action sequences of a given timed transition table (or of a timed automaton) analytically. In the definition of a set of action sequences, when this improves readability and clarity, we write an action sequence $(\sigma_0, t_0)(\sigma_1, t_1) \cdots$ as $(\overline{\sigma}, \overline{t})$ where $\overline{\sigma} = \sigma_0 \, \sigma_1 \cdots$ and $\overline{t} = t_0 \, t_1 \cdots$. The overline notation $\overline{\cdot}$ always denotes a countable infinite sequence of objects in which the positions of the elements are indexed* *by natural numbers. By default the start index is 0. To specify a different start index n we write* $\overline{t}_{i\geq n} = t_n\, t_{n+1} \cdots$ *To specify a finite sequence we use the notation* $\overline{\sigma}_{n\leq i\leq m} = \sigma_n\, \sigma_{n+1} \cdots \sigma_m$ *in which* $m, n \in \mathbb{N}$ *and* $n < m$.

Moreover, we often use ω *and* n *exponentiation as defined in the theory of* ω *languages. We recall the usual conventions in the following:* a^ω *is an infinite string formed by the infinite concatenation of the symbol* a*, while* a^n *is a finite string composed of the concatenation of* n *occurrences of* a*. As usual, if* $n = 0$*, then the result is the empty string* ϵ*. The same convention applies when the base of the exponentiation is a finite string. If the base is a set of symbols* Σ *then the result is a finite or infinite concatenation of symbols of* Σ*.*

For instance, a characterization of the action sequences of T_0 can be written as

$$\begin{aligned} L = \{((ab)^\omega, \overline{t}) \mid \forall i \geq 0.\ t_i \leq t_{i+1} \ \wedge\ t_{2i+1} - t_{2i} \leq 1\} \ \cup \\ \{((ab)^n\, c^\omega, \overline{t}) \mid n \geq 0,\ \forall i \geq 0.\ t_i \leq t_{i+1},\ \forall i. 0 \leq i < 2n \Rightarrow t_{2i+1} - t_{2i} \leq 1, \\ \forall i > 2n.\ t_i = t_{2n} + 2(i - 2n)\} \end{aligned}$$

We can derive the set of repeated states $\mathit{inf}(r)$ of a given fair derivation r. If r is of the form $((ab)^\omega, t_i)$ then we deduce that $\mathit{inf}(r) = \{0, 1\}$. Otherwise, if r is of the form $((ab)^n\, c^\omega, t_i)$, then $\mathit{inf}(r) = \{2\}$.

1.3.3 Zenoness

We want to remark that the associated t.t.s. of a timed transition table could have infinite fair derivations in which the time sequence does not diverge, i.e.

$$\exists \tau \in \mathbb{R}^{>0} \colon \forall i \in \mathbb{N}.\ t_i < \tau$$

This is a consequence of the fact that the time is modeled by real numbers and, thus, the time sequence of a derivation can be a convergent sequence of real numbers. This derivations have been called *Zeno derivations.* They, of course, cannot be considered behaviors of a real-time system and, thus, additional conditions are required on the derivations of the t.t.s. in the context of real-time systems specification and verification. Namely, that every infinite derivation is divergent.

Definition 1.5 (Divergent and Zeno derivations) *Let* $T = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ *be a timed transition table and let* r *be an* infinite *derivation of* $\mathcal{S}(T)$*.* r *is* divergent *iff its time sequence is such that* $\forall \tau \in \mathbb{R}^{>0}. \exists i \in \mathbb{N} \colon t_i > \tau$*. A* Zeno derivation *is a derivation that is not divergent.*

An equivalent characterization requires that in every derivation and any finite interval of time there are finitely many Σ-transitions.

However, Zeno derivations are interesting from the point of view of timed languages theory. In Section 1.4.1 we will treat this aspect in more details.

As an example, a Zeno derivation of the timed transition system $\mathcal{S}(T_0)$ is that in which the sequence of symbols is $(ab)^\omega$ and the time sequence is $t_0 = 0$ and $\forall i > 0.\ t_i = t_{i-1} + \frac{1}{i^2}$. This derivation is possible because the clock constraints of T_0 allows the time between a transition labeled a and the subsequent, labeled b, to be arbitrarily small. On the other hand, if in a derivation at least a transition labeled c is performed, then the clock constraints and the implicit fairness condition imply that the time sequence diverges because the transitions labeled c are separated by exactly 2 time units.

1.3.4 Determinism

Definition 1.1 does not impose any condition on the edges of a timed transition table. Thus, in general, if we consider timed transition tables as automata, they are non-deterministic machines. It is useful, as in the classical automata theory, to identify the subclass of timed transition tables that are deterministic. The presence of time and of the clock constraints onto the edges allow to relax the classical definition: in every state there can be several outgoing edges labeled with the same symbol, provided that their clock constraints forbid the possibility of firing at the same instants.

Definition 1.6 (Deterministic Timed Transition Tables)
Let $T = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ a timed transition table. It is deterministic *if and only if the following conditions hold:*

1. $|B| = 1$, *i.e. there is only one initial state.*

2. $\forall e_1 = (q, \psi_1, \gamma_1, \sigma, q'), e_2 = (q, \psi_2, \gamma_2, \sigma, q'') \in \mathcal{E}.\ \psi_1 \wedge \psi_2$ *is unsatisfiable. In other words, it is required that the clock constraints of any two transitions having the same source and the same label are mutually exclusive.*

As an example, the timed transition table T_0 of Figure 1.2 is deterministic.

1.3.5 Synchronized product

When modeling a system by any formalism, a very useful tool is the possibility to do a modular specification. Different parts of a complex system are specified separately as components and the whole system is specified by a suitable product of all components. Many formalisms introduced to specify reactive systems provide this feature: product of automata, parallel composition of processes in process algebras, product of Petri nets and so on. The possibility of modular specification has been provided also for the specification of real-time systems in all the proposed timed formalisms. A synchronized product is defined between timed transition tables.

An important aspect of the composition operations is the possibility, for different components, to cooperate. For instance, in timed process algebras, such as CCS

[Mil80] or CSP [Hoa78] or ATP [NS94], the parallel composition operator provides a way to the components to synchronize and/or to exchange values by channels. Moreover, each component is always allowed to perform its own operations, i.e. those in which no other component is involved. Then, the whole system behavior is the interleaving of the private operations of the components in parallel and, when certain constraints (which depend on the particular formalism) are satisfied, the common operations of several components synchronizing.

In the timed transition tables (timed automata) formalism, the components are timed transition tables (timed automata) and their product, the whole system, is a timed transition table (timed automaton) too. Its construction is defined according to the following directives.

First, the mechanism of synchronization is based on the common symbols of the alphabets of different components. When a symbol σ is a common symbol, then a transition labeled σ in the whole system can be executed if and only if all components whose alphabet contains σ can execute a transition labeled σ.

Second, in the whole system the time can advance of a certain amount $\delta \in \mathbb{R}^{>0}$ if and only if all the components of the system can let δ time units elapse, according to their structure and semantics.

Finally, in any state of the whole system any component is allowed to perform a private transition (i.e. a transition labeled with a symbol that does not belong to the alphabet of any other component) at any instant in which the transition would have been performed if the component were considered as a stand-alone timed transition table.

Summarizing, the behavior of the whole system should be the interleaving of the behaviors of its components where actions with the same name are executed synchronously.

The formal definition of the whole system, with the characteristics outlined above, is given by syntactically constructing a timed transition table from the specifications of the components.

Definition 1.7 (Synchronized Product) *Let $T_1 = (Q_1, \Sigma_1, \mathcal{E}_1, B_1, \mathcal{X}_1)$ and $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, B_2, \mathcal{X}_2)$ be two timed transition tables with $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. The synchronized product of T_1 and T_2, denoted by $T_1 \mid T_2$, is the following timed transition table:*

$$T_1 \mid T_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \mathcal{E}, B_1 \times B_2, \mathcal{X}_1 \cup \mathcal{X}_2)$$

where $\mathcal{E}$ is defined by:

1. **Synchronization actions**

For all $\sigma \in \Sigma_1 \cap \Sigma_2,\ (q_1, \psi_1, \gamma_1, \sigma, q_1') \in \mathcal{E}_1,\ (q_2, \psi_2, \gamma_2, \sigma, q_2') \in \mathcal{E}_2$

$\mathcal{E}$ *contains* $((q_1, q_2), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2'))$

2. T_1 **actions**

For all $\sigma \in \Sigma_1 \backslash \Sigma_2,\ (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_1,\ s \in Q_2$

$\mathcal{E}$ *contains* $((q, s), \psi, \gamma, \sigma, (q', s))$

3. T_2 **actions**

For all $\sigma \in \Sigma_2 \backslash \Sigma_1,\ (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_2,\ s \in Q_1$

$\mathcal{E}$ *contains* $((s, q), \psi, \gamma, \sigma, (s, q'))$

T_1 *and* T_2 *are called* components *of the synchronized product.*

Let $T_1, T_2, \cdots, T_m, m > 2$*, be* m *timed transition tables such that* $\bigcap_{i=1}^{m} \mathcal{X}_i = \emptyset$ *where* $\mathcal{X}_i$ *is the set of clocks of* T_i*. The synchronized product of them, denoted by* $T_1 \mid T_2 \mid \ldots \mid T_m$*, is defined incrementally starting from the definition above:*

$$T_1 \mid T_2 \mid \ldots \mid T_m = (((\cdots (T_1 \mid T_2) \mid T_3) \mid \cdots) \mid T_m)$$

if the left association rule[1] *is considered.*

This definition shows what we expect in parallel behaviors:

- Common symbols of the alphabets of the components are synchronization actions. A synchronization transition can be executed if and only if all the involved components can execute it (its clock constraints is the conjunction of the clock constraints of the transitions of all components involved). The transition must be executed synchronously by all of the involved components.

- Other symbols can be executed by each component independently according to its original specification.

As an example, consider the timed transition table T_{count} in Figure 1.3. It specifies a binary counter of the occurrences of the symbol b. State 0 represents the case in which the counter value is 0, while state 1 represents the value 1. The system displays, at every time unit, the value of the counter (the corresponding edges are traversed when the value of clock x is equal to 1 and the clock is reset). The symbols `display0` and `display1` model the actions of displaying 0 or 1 respectively. When an action b occurs, the automaton changes its internal state.

The timed transition table $T_{>2}$ in Figure 1.4 simply executes b actions imposing that each of them occurs when at least two time units have elapsed from the previous one.

Now consider the synchronized product $T_{count} \mid T_{>2}$ showed in Figure 1.5. The action b is a common symbol, thus T_{count} and $T_{>2}$ synchronize on it. As a consequence, in the whole system the counter cannot change its internal state at any time as in T_{count}, because the time between two subsequent occurrences of b must be greater than 2.

[1]Equivalently, the right association rule can be used. As a matter of fact, the synchronized product operation is associative.

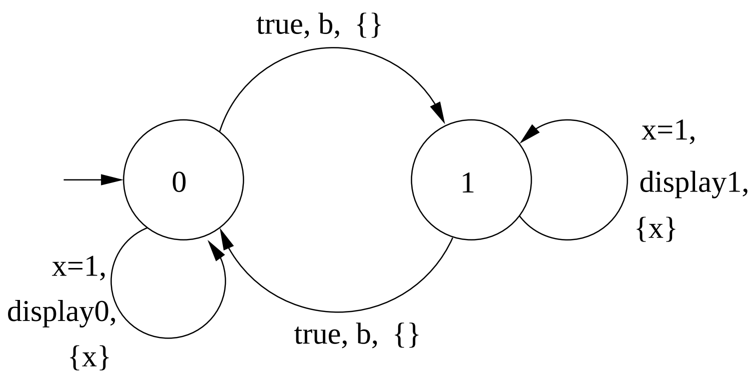

Figure 1.3: A simple binary counter, T_{count}

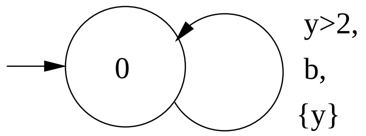

Figure 1.4: A timed transition table $T_{>2}$

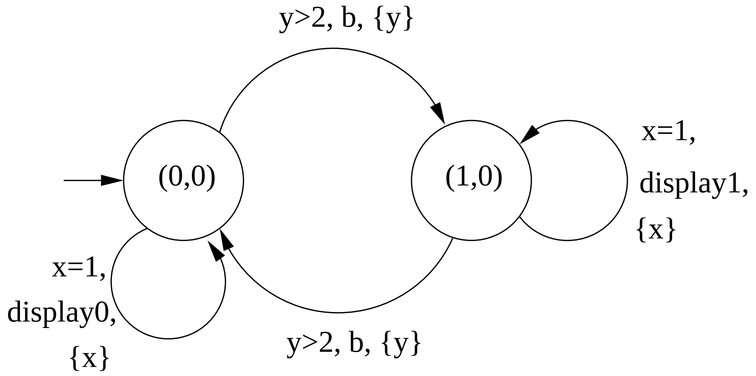

Figure 1.5: The timed transition table $T_{count} \mid T_{>2}$

1.4 Timed Automata

As we mentioned in Section 1.3, timed transition tables are the basic objects on which different types of timed automata can be defined. In particular, timed transition tables can be considered the "greatest common divisor" of several formalisms that are all referred to as timed automata. In particular we think of Timed Büchi Automata and Timed Muller Automata of [AD94] together with the numerous variants and extensions of these models that have been proposed in the literature. In this section we introduce formally the models of [AD94].

1.4.1 Büchi acceptance condition

The theory of ω-languages is the extension of the classical formal languages theory on finite words to the study of languages composed by infinite words. Let Σ be a finite alphabet of symbols. As usual, Σ^* denotes the set of all finite words composed by symbols in Σ and Σ^∞ denotes the set of all infinite words composed by symbols of Σ. As formal languages are subsets of Σ^*, ω-languages are subsets of $\Sigma^* \cup \Sigma^\infty$. ω-regular-languages are those ω-languages that can be accepted by finite automata together with accepting conditions for infinite words (ω-automata) [B̈60, Mul63]. [AD94] exposes a theory of timed regular languages starting from the theory of ω-regular-languages.

Definition 1.8 (Timed Büchi Automaton) *A timed Büchi automaton (TBA) is a tuple $T = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ where $(Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ is a timed transition table and $R \subseteq Q$, $R \neq \emptyset$, is a set of* repeated states*.*
The timed transition table of a TBA T is denoted by $\widehat{T} = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$.
The class of all TBAs is denoted by $\mathfrak{T}$.
If $\widehat{T}$ is a deterministic timed transition table (see Definition 1.6), then we say that T is a deterministic timed Büchi automaton. The class of all deterministic TBAs is denoted by $\mathfrak{T}_{\mathfrak{D}}$.

The set of repeated states is needed to specify a Büchi acceptance condition for infinite timed words [B̈60].

Definition 1.9 (Timed Word, Timed Language) *Let Σ be a finite alphabet. A* timed word *over Σ is a finite or infinite sequence of pairs $(\sigma_0, t_0)(\sigma_1, t_1)\cdots$ such that $\forall i.\ \sigma_i \in \Sigma \ \wedge\ t_i \in \mathbb{R}^{\geq 0} \ \wedge\ t_i \leq t_{i+1}$.*
A timed language *over Σ is a subset of the set of all timed words over Σ.*

In [AD94] and many other papers based on the model specified there, the emphasis is posed only on infinite timed words because the aim of the theory is to provide a framework for the study of real-time systems. However, in some papers that focus on the theory of timed regular languages and that report attempts to give timed versions of classical theorems and properties of formal languages theory, also finite timed words are considered [BPDG98, ACM97, ACM02, BP99, BPT03]. In this thesis we always focus our attention to infinite timed words unless finite timed words are explicitly mentioned.

Moreover, also non-divergent words are not considered as semantics of timed automata in the framework of specification and verification of real-time system. For this reason, several methods have been proposed to get rid of these words, especially in the model checking area (see Section 2.3.4). Let us fix some conventional terms:

Definition 1.10 (Divergent and Zeno words) *Let Σ be a finite alphabet of symbols. An infinite timed word over Σ, $w = (\sigma_0, t_0)(\sigma_1, t_1)\cdots$, is a* divergent *word iff $\forall \tau \in \mathbb{R}^{\geq 0}$. $\exists i \in \mathbb{N}$: $t_i > \tau$. If w is not divergent, then it is a* Zeno *word, i.e. $\exists \gamma \in \mathbb{R}^{\geq 0}$: $\forall i \in \mathbb{N}$. $t_i < \gamma$.*

Convention 6 *Timed Büchi Automata on infinite timed words are considered, in this thesis, the underlying model. For this reason the term "timed automaton" is used as a synonym for "timed Büchi automaton". Note that timed automata are ranged over by $T, T', T_1, T_2, \ldots$ which is the meta-notation used also for timed transition tables. This should not create confusion because, generally, the type of object to which we are referring is clear from the context. When this abuse of notation could create confusion, the notation $\widehat{T}$, introduced in Definition 1.8, will be used to distinguish a timed transition table from a timed automaton.*

Definition 1.11 (Run, Acceptance) *Let $T = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton and let $\widehat{T}$ be its timed transition table. An infinite derivation r of the transition system $\mathcal{S}(\widehat{T})$ is called a* **run** *of T if and only if the following conditions hold:*

1. *r is a fair derivation.*
2. *$inf(r) \cap R \neq \emptyset$.*

A timed word w over Σ is **accepted** *by T if a run r of T exists such that $w = v$, where v is the action sequence of r.*
The set of all timed words accepted by T is called the **accepted language** *of T and it is denoted by $\mathcal{L}(T)$.*

The Büchi acceptance condition is a further fairness condition imposed on fair derivations of $\mathcal{S}(\widehat{T})$. Only those derivations that satisfy the two fairness conditions are considered behaviors of the system, i.e. timed words accepted by the timed automaton T. However, note that if the set of repeated states R is equal to Q we have that T accepts all the timed words originated from all fair derivations of $\mathcal{S}(\widehat{T})$.

Consider again the timed transition table T_0 of Figure 1.2. Let us define a timed automaton based on T_0 adding, as set of repeated states, the set $R_0 = \{2\}$. In

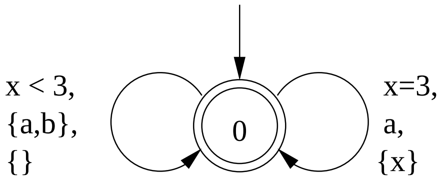

Figure 1.6: Timed automaton T_{3a}

Section 1.3.2 the set of all action sequences of T_0 has been characterized analytically. This example explains very well the impact of the further fairness condition imposed by the Büchi acceptance condition. The fact that the unique repeated state is the state 2 discards a lot of actions sequences deriving from fair derivations, namely those that never enter state 2 and that cycle forever between the states 0 and 1. Thus, the timed language accepted by T_0 with 2 as unique repeated state is

$$\mathcal{L}(T_0) = \{((ab)^n\, c^\omega, \overline{t}) \mid n \geq 0,\ \forall i \geq 0.\ t_i \leq t_{i+1},\ \forall i. 0 \leq i < 2n \Rightarrow t_{2i+1} - t_{2i} \leq 1,\\ \forall i > 2n.\ t_i = t_{2n} + 2(i - 2n)\}$$

Convention 7 *Note that we use the same conventions used to specify action sequences, which, indeed, are timed words. Throughout the thesis we use the usual graphical notation for specifying repeated states: they are depicted as double circles.*

The use of Büchi acceptance conditions is, from the point of view of the specification of real-time systems, a powerful tool because typical requirements can be expressed in this way. For instance, giving the set $R_0 = \{2\}$ in the previous example corresponds to specify that the real-time system modeled by the timed automaton *eventually* enters state 2.

Let us give another example of timed automaton. Figure 1.6 shows an automaton T_{3a} in which $\Sigma = \{a, b\}$ and $Q = B = R = \{0\}$. In this case, as in all cases in which $R = Q$, the Büchi acceptance condition is useless, i.e. the action sequence of *every* fair derivation of $\widehat{T_{3a}}$ is an accepted word of T_{3a}. The language is

$$\mathcal{L}(T_{3a}) = \{(\overline{\sigma}, \overline{t}) \cdots \mid \forall i \in \mathbb{N}.\ \sigma_i \in \{a, b\},\ \forall i \in \mathbb{N}^{>0}.\ \exists j\colon\ t_j = 3i\ \wedge\ \sigma_i = a\}$$

1.4.2 Parallel composition

The product of timed transition tables is the first step toward the definition of the parallel composition between timed automata. As standard parallel composition operators, the parallel composition of timed automata is defined requiring the following fairness condition: in every run r of the obtained automaton, the projection of r on each component j results in a derivation which is a run of the j-th automaton. Thus, the parallel composition automaton has to be defined such that an infinite derivation of its timed transition table is a run if and only all the correspondent derivations of the components satisfy the Büchi acceptance condition according to their sets of repeated states.

This can be done, as outlined in [AD94, Alu99], by adding a number to each state of the parallel composition automaton. This number behaves as a counter that keeps track of which components entered one of their repeated states. Consider two timed automata $T_1 = (Q_1, \Sigma_1, \mathcal{E}_1, B_1, R_1, \mathcal{X}_1)$ and $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, B_2, R_2, \mathcal{X}_2)$. The set of states of their parallel composition is $Q_1 \times Q_2 \times \{0, 1, 2\}$. Initially the counter is 0. If the first component enters a state belonging to R_1 by a transition, the new state have the third component equal to 1. Then, when the second component will enter a state belonging to R_2 the new state will have the counter at 2. The subsequent transition resets the counter to 0.

Then, the right definition is achieved by requiring that the runs of the parallel composition automaton are those in which the counter assume the value 2 infinitely many times.

Definition 1.12 (Parallel Composition) *Let* $T_1 = (Q_1, \Sigma_1, \mathcal{E}_1, B_1, R_1, \mathcal{X}_1)$ *and* $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, B_2, R_2, \mathcal{X}_2)$ *be two timed automata such that* $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$.
The parallel composition *of* T_1 *and* T_2, *denoted by* $T_1 \parallel T_2$, *is the following timed automaton:*

$$T_1 \parallel T_2 = (Q_1 \times Q_2 \times \{0,1,2\}, \Sigma_1 \cup \Sigma_2, \mathcal{E}, B_1 \times B_2 \times \{0\}, Q_1 \times Q_2 \times \{2\}, \mathcal{X}_1 \cup \mathcal{X}_2)$$

where $\mathcal{E}$ *is defined as follows:*

1. **Synchronization actions**

For all $\sigma \in \Sigma_1 \cap \Sigma_2$, $(q_1, \psi_1, \gamma_1, \sigma, q_1') \in \mathcal{E}_1$, $(q_2, \psi_2, \gamma_2, \sigma, q_2') \in \mathcal{E}_2$

$\mathcal{E}$ *contains:*

- $((q_1, q_2, i), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 2))$ *iff* $q_1' \in R_1$ *and* $q_2' \in R_2$ *where* $i \in \{0, 1, 2\}$
- $((q_1, q_2, i), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', i))$ *iff* $q_1' \notin R_1$ *and* $q_2' \notin R_2$ *where* $i \in \{0, 1\}$
- $((q_1, q_2, 2), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 0))$ *iff* $q_1' \notin R_1$ *and* $q_2' \notin R_2$
- $((q_1, q_2, 0), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 0))$ *iff* $q_1' \notin R_1$ *and* $q_2' \in R_2$
- $((q_1, q_2, 1), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 2))$ *iff* $q_1' \notin R_1$ *and* $q_2' \in R_2$
- $((q_1, q_2, 2), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 0))$ *iff* $q_1' \notin R_1$ *and* $q_2' \in R_2$

- $((q_1, q_2, 0), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 1))$ *iff* $q_1' \in R_1$ *and* $q_2' \notin R_2$
- $((q_1, q_2, 1), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 1))$ *iff* $q_1' \in R_1$ *and* $q_2' \notin R_2$
- $((q_1, q_2, 2), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2', 1))$ *iff* $q_1' \in R_1$ *and* $q_2' \notin R_2$

2. T_1 **actions**

   For all $\sigma \in \Sigma_1 \backslash \Sigma_2,\ (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_1,\ s \in Q_2$

   $\mathcal{E}$ *contains:*

   - $((q, s, 1), \psi, \gamma, \sigma, (q', s, 1))$
   - $((q, s, 0), \psi, \gamma, \sigma, (q', s, 0))$ *iff* $q' \notin R_1$
   - $((q, s, 2), \psi, \gamma, \sigma, (q', s, 0))$ *iff* $q' \notin R_1$
   - $((q, s, 0), \psi, \gamma, \sigma, (q', s, 1))$ *iff* $q' \in R_1$
   - $((q, s, 2), \psi, \gamma, \sigma, (q', s, 1))$ *iff* $q' \in R_1$

3. T_2 **actions**

   For all $\sigma \in \Sigma_2 \backslash \Sigma_1,\ (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_2,\ s \in Q_1$

   $\mathcal{E}$ *contains:*

   - $((s, q, 2), \psi, \gamma, \sigma, (s, q', 0))$
   - $((s, q, 0), \psi, \gamma, \sigma, (s, q', 0))$
   - $((s, q, 1), \psi, \gamma, \sigma, (s, q', 1))$ *iff* $q' \notin R_2$,
   - $((s, q, 1), \psi, \gamma, \sigma, (s, q', 2))$ *iff* $q' \in R_2$

Let $T_1, T_2, \cdots, T_m, m > 2$, *be* m *timed automata such that* $\bigcap_{i=1}^{m} \mathcal{X}_i = \emptyset$ *where* $\mathcal{X}_i$ *is the set of clocks of* T_i. *The parallel composition of them, denoted by* $T_1 \parallel T_2 \parallel \cdots \parallel T_m$, *is defined incrementally starting from the definition above:*

$$T_1 \parallel T_2 \parallel \ldots \parallel T_m = (((\cdots (T_1 \parallel T_2) \parallel T_3) \parallel \cdots) \parallel T_m)$$

if the left association rule[2] *is considered.*

Definition 1.13 (Projections) *Let* $T_1 = (Q_1, \Sigma_1, \mathcal{E}_1, B_1, R_1, \mathcal{X}_1)$ *and* $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, B_2, R_2, \mathcal{X}_2)$ *be two timed automata such that* $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$.
Let $r = ((q_0^1, q_0^2, k_0), \nu_0^1 \cup \nu_0^2) \xrightarrow{l_0} ((q_1^1, q_1^2, k_1), \nu_1^1 \cup \nu_1^2) \xrightarrow{l_1} \cdots$ *be an infinite derivation of the timed transition table* $\widehat{T_1 \parallel T_2}$

[2]Equivalently, the right association rule can be used. As a matter of fact, the parallel composition operation is associative up to simple isomorphisms between states.

where for all $i \in \mathbb{N}$, $q_i^1 \in Q_1, q_i^2 \in Q_2, \nu_i^1 \in \mathcal{V}_{\mathcal{X}_1}, \nu_i^2 \in \mathcal{V}_{\mathcal{X}_2}$ *and* $k_i \in \{0,1,2\}$
The projection *of* r *on the component* T_1 *is the infinite derivation* $r_{|1}$ *obtained by* r *in the following way:*

- *the initial state* $((q_0^1, q_0^2, 0), \nu_0^1 \cup \nu_0^2)$ *is transformed into* (q_0^1, ν_0^1)
- *all parts of the form* $\xrightarrow{l_{i-1}}((q_i^1, q_i^2, k_i), \nu_i^1 \cup \nu_i^2)$ *such that* $i \geq 1$ *and* $l_{i-1} \in \Sigma_2 \backslash \Sigma_1$ *are deleted*
- *all remaining parts of the form* $\xrightarrow{l_{i-1}}((q_i^1, q_i^2, k_i), \nu_i^1 \cup \nu_i^2)$ *are transformed into* $\xrightarrow{l_{i-1}}(q_i^1, \nu_i^1)$

The projection *of* r *on the component* T_2*, denoted by* $r_{|2}$*, is obtained symmetrically.*

By a projection we retain all δ-transitions and those transitions in which the component we are projecting on is involved.

Proposition 1.1 (Fairness of the $\|$ operator) *Let* $T_1 = (Q_1, \Sigma_1, \mathcal{E}_1, B_1, R_1, \mathcal{X}_1)$ *and* $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, B_2, R_2, \mathcal{X}_2)$ *be two timed automata such that* $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$*. Let* r *be a run of the timed automaton* $T_1 \parallel T_2$*. Then,* $r_{|1}$ *is a run of* T_1 *and* $r_{|2}$ *is a run of* T_2*.*

Proof. Let r be such a run. We start showing that $r_{|1}$ can be derived by $\mathcal{S}(\widehat{T_1})$ starting from an initial state.

By definition, to obtain $r_{|1}$, we retain all δ-transitions and those transitions in which T_1 participates, i.e. those of the form $((q_{i-1}^1, q_{i-1}^2, k_{i-1}), \nu_{i-1}^1 \cup \nu_{i-1}^2) \xrightarrow{\sigma_{i-1}}((q_i^1, q_i^2, k_i), \nu_i^1 \cup \nu_i^2)$ where $\sigma_{i-1} \in \Sigma_1$ can be a synchronization action or an action of T_1.

Such transition is performed by $\mathcal{S}(\widehat{T_1 \parallel T_2})$ using the rule T2 of Figure 1.1. Thus, in $\widehat{T_1 \parallel T_2}$, there exists an edge $((q_{i-1}^1, q_{i-1}^2, k_{i-1}), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma_{i-1}, (q_i^1, q_i^2, k_i))$ where $\psi_j \in \Psi_{\mathcal{X}_j}$ and $\gamma_j \in \Gamma_{\mathcal{X}_j}$ and $j \in \{1,2\}$[3].

By the structure of $\widehat{T_1 \parallel T_2}$ specified in Definition 1.12, this implies that

1. there exists an edge in $\mathcal{E}_1$ of the form $((q_{i-1}^1, q_{i-1}^2), \psi_1, \gamma_1, \sigma_{i-1}, (q_i^1, q_i^2))$
2. $\nu_{i-1}^1 \models \psi_1$
3. $\nu_i^1 = \nu_{i-1}^1 \backslash \gamma_1$

Thus, starting from an initial state of $\mathcal{S}(\widehat{T_1})$ the derivation $r_{|1}$ can be constructed using the rules of Figure 1.1 (note that δ-transitions have not premises to satisfy other than $\delta > 0$, which is obviously true in all cases).

[3]When σ_{i-1} is not a synchronization action we set $\psi_2 = true$ and $\gamma_2 = \emptyset$.

The same argumentation can be used to prove that $r_{|2}$ can be derived in $\mathcal{S}(\widehat{T_2})$.

Now we show that $r_{|1}$ is a fair derivation of $\widehat{T_1}$ and that $inf(r_{|1}) \cap R_1 \neq \emptyset$. This is equivalent to say that the state sequence of $r_{|1}$ is infinite and there exists a state in R_1 that appears infinitely often in it.

The fact that r is a run implies that its state sequence $\overline{q_r} = (q_0^1, q_0^2, k_0)(q_1^1, q_1^2, k_1)\cdots$ is infinite and that there exists a state $(\widetilde{q^1}, \widetilde{q^2}, 2) \in \{(q, q', 2) \mid q \in Q_1,\ q' \in Q_2,\}$ that occurs infinitely often in it.

By construction of $\widehat{T_1 \parallel T_2}$, that uses the counters in the states specified in Definition 1.12, we know that if $(\widetilde{q^1}, \widetilde{q^2}, 2)$ occurs at a certain position of $\overline{q_r}$, then, during r and before this occurrence of the state $(\widetilde{q^1}, \widetilde{q^2}, 2)$ is entered, each component has performed a transition whose target state is one of its repeated states.

Thus, in r, before the considered occurrence of the state $(\widetilde{q^1}, \widetilde{q^2}, 2)$ is entered, there is at least one transition of the form $((q_{i-1}^1, q_{i-1}^2, k_{i-1}), \nu_{i-1}^1 \cup \nu_{i-1}^2) \xrightarrow{\sigma_{i-1}} ((q_i^1, q_i^2, k_i), \nu_i^1 \cup \nu_i^2)$ where $\sigma_{i-1} \in \Sigma_1$ is a synchronization action (or an action of T_1) and $q_i^1 \in R_1$.

We know that this transition is retained to obtain $r_{|1}$ and it becomes a transition of the form $(q_{i-1}^1, \nu_{i-1}^1) \xrightarrow{\sigma_{i-1}} ((q_i^1, \nu_i^1)$ at some position of $r_{|1}$. Thus, a repeated state q_i^1 occurs at a certain position in the state sequence of $r_{|1}$ given an occurrence of the state $(\widetilde{q^1}, \widetilde{q^2}, 2)$ in $\overline{q_r}$.

We are supposing that $(\widetilde{q^1}, \widetilde{q^2}, 2)$ occurs infinitely often in $\overline{q_r}$ and now we show that, for every such occurrence, there is a different occurrence of a repeated state of R_1 in the state sequence of $r_{|1}$. We use an inductive argument. The previous discussion shows that, for the first occurrence of $(\widetilde{q^1}, \widetilde{q^2}, 2)$ in $\overline{q_r}$, there is a repeated state of T_1 in $r_{|1}$. By construction, the counter 2 in the state $(\widetilde{q^1}, \widetilde{q^2}, 2)$ is reset to 0 in the subsequent $(\Sigma_1 \cup \Sigma_2)$-transition of r. Thus, a subsequent occurrence of $(\widetilde{q^1}, \widetilde{q^2}, 2)$ in $\overline{q_r}$ implies that, in the meanwhile, another different transition of the form $((q_{j-1}^1, q_{j-1}^2, k_{j-1}), \nu_{j-1}^1 \cup \nu_{j-1}^2) \xrightarrow{\sigma_{j-1}} ((q_j^1, q_j^2, k_j), \nu_j^1 \cup \nu_j^2)$ with $q_j^1 \in R_1$ has occurred. Consequently, another different occurrence of a repeated state of T_1 has occurred in the state sequence of $r_{|1}$. This argument can be iterated for every subsequent occurrence of $(\widetilde{q^1}, \widetilde{q^2}, 2)$ in $\overline{q_r}$.

Thus, there is an infinite number of occurrences of states of R_1 in the state sequence of $r_{|1}$. Since R_1 is a finite set, there must be a certain $q \in R_1$ that occurs infinitely often in the state sequence of $r_{|1}$. This implies, by definition, that $r_{|1}$ is a run of T_1.

It can be proved in the same way that $r_{|2}$ is a run of T_2. ■

Let us show an example of construction of parallel composition using the previous definition. We show also that the simple synchronized product of the timed transition tables of two timed automata cannot be used as a timed transition table for the parallel composition of them.

Figures 1.7 and 1.8 show two timed automata T_1 and T_2 such that:

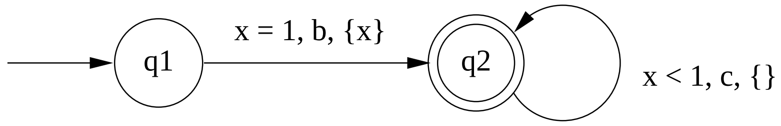

Figure 1.7: Automaton T_1

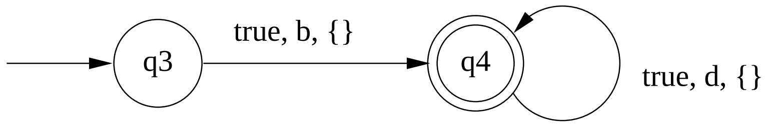

Figure 1.8: Automaton T_2

$$\mathcal{L}(T_1) = \{(b\,c^\omega, \bar{t}) \mid t_0 = 1, \forall i \in \mathbb{N}.\ t_i \leq t_{i+1},\ \exists \gamma \in \mathbb{R}^{>0}:\ 1 < \gamma < 2\ \wedge\ \forall i \in \mathbb{N}.\ t_i < \gamma\}$$

$$\mathcal{L}(T_2) = \{(b\,d^\omega, \bar{t}) \mid \forall i \in \mathbb{N}.\ t_i \leq t_{i+1}\}$$

Note that $\mathcal{L}(T_1)$ contains only Zeno words, while $\mathcal{L}(T_2)$ contains both divergent and Zeno words.

The synchronized product $\widehat{T_1} \mid \widehat{T_2}$ is showed in Figure 1.9.

Convention 8 *The definition of synchronized product and of parallel composition is given in term of set of states which are the Cartesian product of the states of the components. When representing graphically the result, we, as usual, depict only the states and the edges of the "reachable" part, i.e. the part of the graph that can be reached by a path, along directed edges, starting at an initial state.*

It is easy to see that we cannot find a set of repeated states for $\widehat{T_1} \mid \widehat{T_2}$ in order to obtain a timed automaton in which every run can be projected on T_1 and T_2 runs. To see this, let us try all possibilities. First of all, note that the state (q_1, q_3) is leaved by the first transition of every run and it is not entered by any transition.

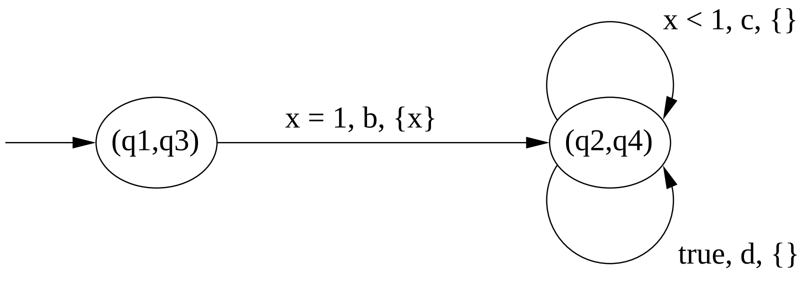

Figure 1.9: Timed transition table $\widehat{T_1} \mid \widehat{T_2}$

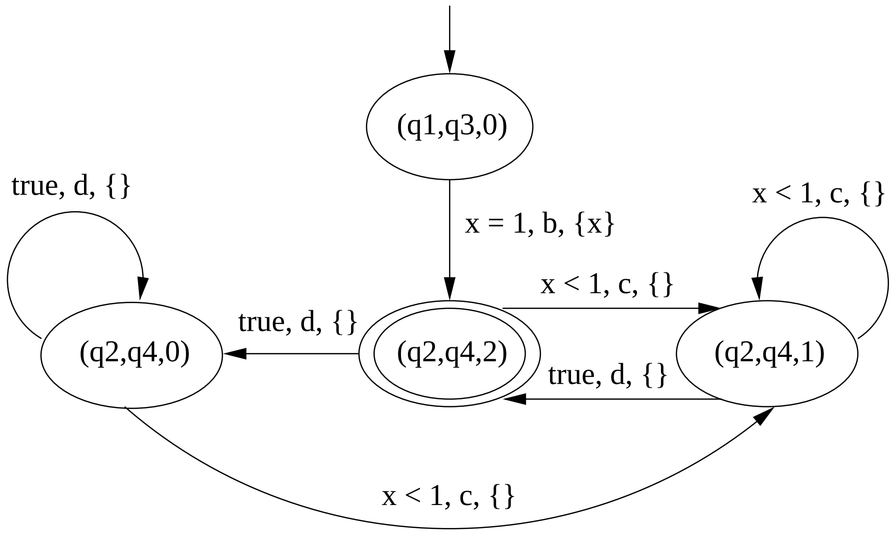

Figure 1.10: Timed automaton $T_1 \parallel T_2$

Thus, making this state a repeated state is useless because it can never be entered infinitely many times in any run. The only other possibility is that the repeated state is (q_2, q_4).

This choice is not correct because we have that the resulting timed automaton accepts timed words in the set $L_{err} = \{(b\,\{c,d\}^n\,d^\omega, \bar{t}) \mid t_0 = 1, \forall i.\ 1 \le i < n \Rightarrow 1 \le t_i \le t_{i+1} < 2,\ t_n < 2,\ t_n \le t_{n+1},\ \forall i.\ n < i \Rightarrow t_i \le t_{i+1}\}$. These timed words are accepted by runs that, projected on the component T_1, are finite derivations, which are *not* runs of T_1.

Thus, we cannot find, in general, a way to transform $\widehat{T_1} \mid \widehat{T_2}$ into a parallel composition of T_1 and T_2. Consequently, the construction of Definition 1.12 that uses counters in the states is needed if we want to define fair parallel compositions.

Figure 1.10 shows the timed automaton $T_1 \parallel T_2$. We have, correctly, that:

$$\mathcal{L}(T_1 \parallel T_2) = \{(b\,\{c,d\}^\omega, \bar{t}) \mid t_0 = 1, \forall i \in \mathbb{N}^{>0}.\ t_i \le t_{i+1} < 2\}$$

Note that the parallel composition force T_2 to accept only Zeno words, as T_1. We want to remark that the impossibility to obtain the parallel composition from the graph of $\widehat{T_1} \mid \widehat{T_2}$ does not depend on the fact that one or both the components accept only Zeno words. Indeed, if we delete the constraint $x < 1$ in T_1, then both the components can accepts divergent timed words, but the words of L_{err} still can be accepted.

1.4.3 Muller Acceptance condition

The Büchi acceptance condition is not the only proposed device to specify how infinite words can be accepted by a finite automaton. We recall the Muller acceptance condition [Mul63] because it is used in [AD94] to define classes of timed automata with interesting characteristics.

Definition 1.14 (Timed Muller Automaton) *A timed Muller automaton T consists of a timed transition table $\widehat{T} = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ together with a family of sets of accepting states $\mathcal{F} \subseteq \wp(Q)$.*

The class of all timed Muller Automata is denoted by $\mathcal{M}$.

If the timed transition table $\widehat{T}$ is deterministic (see Definition 1.6) then we say that T is a deterministic timed Muller automaton. The class of all deterministic timed Muller automata is denoted by $\mathcal{M}_{\mathcal{D}}$.

Definition 1.15 (Run, Acceptance)
Let $T = (Q, \Sigma, \mathcal{E}, B, \mathcal{F}, \mathcal{X})$ be a timed Muller automaton and let $\widehat{T}$ be its timed transition table. An infinite derivation r of the transition system $\mathcal{S}(\widehat{T})$ is called a **run** *of T if and only if the following conditions hold:*

1. *r is a fair derivation.*

2. *$inf(r) \in \mathcal{F}$.*

A timed word w over Σ is **accepted** *by T if a run r of T exists such that $w = v$, where v is the action sequence of r.*

The set of all timed words accepted by T is called the **accepted language** *of T and it is denoted by $\mathcal{L}(T)$.*

The Muller acceptance condition correspond to a different way of specifying fairness conditions on infinite derivations. In the following section we show some results that relate the class of timed automata that we have introduced so far.

1.5 Equivalence and Expressive Power

Indeed, given a timed language L, different automata could exist that accept L. This leads to a natural notion of equivalence between timed automata.

Definition 1.16 (Trace Equivalence) *Let T_1 and T_2 be two timed automata. We say that they are equivalent, with respect to their timed traces or with respect to accepted timed languages, if and only if the following condition holds.*

$$\mathcal{L}(T_1) = \mathcal{L}(T_2)$$

If T_1 and T_2 are equivalent in this way we denote this fact by $T_1 \approx T_2$.

When we have different classes of automata, the notion of trace equivalence can be used to relate them with respect to the so-called *expressive power*.

Definition 1.17 (Expressive Power)
Let $\mathcal{C}$ be a class of automata accepting timed languages. By $\mathcal{L}(\mathcal{C})$ we denote the set of timed languages that can be accepted by the automata in the class $\mathcal{C}$, i.e. the set of timed languages L such that there exists an automaton T in the class $\mathcal{C}$ with $\mathcal{L}(T) = L$.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two classes of automata accepting timed languages. We say that the class $\mathcal{C}_1$ has the same expressive power *of the class $\mathcal{C}_2$ if and only if $\mathcal{L}(\mathcal{C}_1) = \mathcal{L}(\mathcal{C}_2)$.*

The expressive power of $\mathcal{C}_1$ is less than that of $\mathcal{C}_2$ if and only if $\mathcal{L}(\mathcal{C}_1) \subset \mathcal{L}(\mathcal{C}_2)$. The expressive power of $\mathcal{C}_1$ is greater than that of $\mathcal{C}_2$ if and only if $\mathcal{L}(\mathcal{C}_1) \supset \mathcal{L}(\mathcal{C}_2)$.

The notion of expressive power is a significant parameter to consider when relating different classes of automata especially when new classes of automata are defined as extensions of existing ones. In the following, when presenting an extension or a variant of the class of timed automata, we always analyze the expressive power of the new class with respect to the original class.

For instance, in [AD94], the authors show the following relations among the classes $\mathcal{T}$, $\mathcal{T}_\mathcal{D}$, $\mathcal{M}$ and $\mathcal{M}_\mathcal{D}$:

$$\mathcal{L}(\mathcal{M}_\mathcal{D}) \subset \mathcal{L}(\mathcal{T}_\mathcal{D}) \subset \mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{T})$$

Note that the last equality means that we can simulate a Muller acceptance condition with a Büchi acceptance condition by a suitable construction (and also the vice-versa) when considering non-deterministic timed transition tables. For this reason we, following the choice of [AD94], have used the Büchi acceptance condition in all the definitions: it is simpler to manage than the other and have the same expressive power.

Moreover, this results show a first difference between the classical theory of regular languages and the theory of timed regular languages. Namely, that the deterministic and non-deterministic automata have different expressive power. This is true, in a different way, also for the theory of ω-regular-languages in which only deterministic Büchi automata have less expressive power than others.

This is not the only difference between untimed regular languages and timed ones. In Section 1.7 we show another important difference regarding non-observable transitions.

1.6 Normal forms of Timed Automata

In this section we show several constructions and restrictions that can be applied to a timed automaton obtaining an equivalent one that has a simplified form.

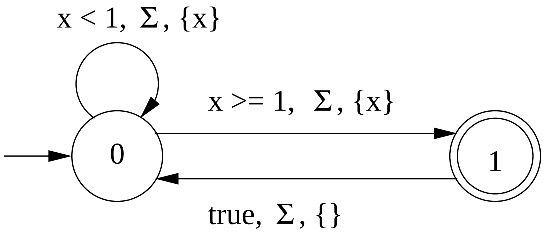

Figure 1.11: Automaton T_{NZ}

1.6.1 Restriction to Integer Constants

In Section 1.2 we have defined clock constraints in such a way that the constants to which the clocks are compared are non-negative rational numbers. However, in the fundamental constructions for the verification of timed automata it is required to have integer constants (see Section 2.2.1).

This can be simply obtained multiplying every rational constant of the set C of constants appearing in the constraints of a timed automaton T by the greatest common multiple of the denominators of the numbers in C. In this way we obtain a timed automaton that is isomorphic to T.

The difference between the two automata is essentially in the time unit of measure used. When the timed automaton is used to model a real-time system this difference is not influent: rational constants can be used in the design task and then, at the verification task, the constants can be transformed into integers as above. All the properties of the original automaton are preserved.

1.6.2 Avoiding Zeno Runs

In the task of modeling a real-time system by a timed automaton one would like to rule out the possible Zeno runs because they do not model a real behavior of the system. One way to do this task is to use the timed language accepted by the automaton T_{NZ} of Figure 1.11. This language is

$$L_{NZ} = \{(\overline{\sigma}, \overline{t}) \mid \forall i \in \mathbb{N}.\ \sigma \in \Sigma \wedge t_i \leq t_{i+1},\ \forall \tau \in \mathbb{R}^{>0}.\exists i \in \mathbb{N}\colon\ t_i > \tau\}$$

Given a timed automaton T having the set Σ as alphabet of symbols, the parallel composition $T \parallel T_{NZ}$ is an automaton accepting all the timed words accepted by T such that their time sequence is divergent.

However, this operation is not needed in the verification task because the verification algorithms often are designed to consider only the divergent runs in checking the properties.

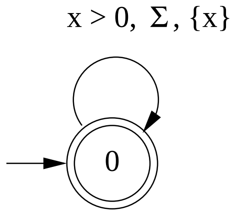

Figure 1.12: Automaton T_{Strict}

1.6.3 Strictly increasing time sequence

By our definition of the derivations of the timed transition system $\mathcal{S}(\widehat{T})$ of a timed transition table (Definition 1.3) we have that, in general, the time sequence of any derivation is such that $\forall i \in \mathbb{N}.\ t_i \leq t_{i+1}$.

However, in several papers (also in [AD94]) it is imposed that the time sequences are strictly increasing: $\forall i \in \mathbb{N}.\ t_i < t_{i+1}$. This corresponds to forbid two Σ-transition to occur at the same time instant. Thus, to model this situations, the powerset $\wp(\Sigma)$ (without the empty set) has to be considered as the alphabet of the automaton and the occurrence of a set of symbols in a certain time instant represents the fact that all symbols in the set occurred in that instant.

All the results about timed automata are true both in the case of strictly increasing time sequences and in the case of non-decreasing ones. We have adopted the latter assumption because it seems more natural and simpler to manage. However, if strictly increasing time sequences are needed, it is possible, as for Zeno runs, to rule out all timed words with a non-decreasing time sequence of an automaton T considering the parallel composition $T \parallel T_{Strict}$. The automaton T_{Strict} is shown in Figure 1.12 and its accepted language is

$$L_{Strict} = \{(\overline{\sigma}, \overline{t}) \mid \forall i \in \mathbb{N}.\ \sigma_i \in \Sigma \wedge t_i < t_{i+1}\}$$

1.6.4 Minimal syntax for clock constraints

In Section 1.2 we have given the syntax rules for clock constraints and we observed that the syntax allows to express different clock constraints having the same meaning, as, for instance, $x = 3$ and $x \leq 3 \wedge x \geq 3$. When proving properties or giving complex constructions of timed automata it is useful to consider a minimal non-redundant set of constraints (see for instance the construction of Chapter 4).

Using classical logical equivalence as De Morgan laws, we can put any clock constraint ψ into disjunctive normal form $\psi_1 \vee \cdots \vee \psi_n$. Then we can simulate the disjunction by non-determinism of timed automata in the following way: the edge $(q, \psi, \gamma, \sigma, q')$ becomes n edges $(q, \psi_1, \gamma, \sigma, q')$, $(q, \psi_2, \gamma, \sigma, q')$, $\cdots$ $(q, \psi_n, \gamma, \sigma, q')$. Applying this transformation to every edge of a timed automaton we obtain a timed automaton that is equivalent to the original one and that is said to be *disjunction free*.

Moreover we can use the properties of the order of real-numbers to simulate the equality, as we outlined in the example above, and we can simulate $true$ ($false$) with $x \geq 0$ ($x < 0$).

At this stage of the analysis we can restrict to the following syntax (we use integer constants):

$$\begin{array}{rcl} \psi & ::= & x \# c \\ & & \mid x - y \# c \\ & & \mid \psi \wedge \psi \\ & & \mid \neg \psi \end{array}$$

where $\# \in \{<, \leq, >, \geq\}$, $c \in \mathbb{N}$ and x, y are clock variables.

Now, note that the negation operator is not needed because the negation of an atomic constraint $x \# t$ can be expressed as another constraint of the same kind. An atomic constraint of the form $x = t$ can be expressed as the conjunction $x \leq t \wedge x \geq t$ and its negation can be expressed by $x < t \vee x > t$. The disjunction, as above, can be simulated by the non-determinism of the edges. Moreover, in [AD94] it is stated that *diagonal constraints*, i.e. those of the form $x - y \# c$, are not needed, in the sense that they can be simulated using the other constraints. In [BPDG98] a construction to eliminate all diagonal constraints from a timed automaton is given. However, the result of this construction has a size significantly greater than that of the original automaton and this has to be considered when assuming to use a *diagonal free* timed automaton.

Putting all together, the minimal non-redundant syntax for clock constraints is the following:

$$\begin{array}{rcl} \psi & ::= & x \# c \\ & & \mid \psi \wedge \psi \end{array}$$

where $\# \in \{<, \leq, >, \geq\}$, $c \in \mathbb{N}$ and x is a clock variable.

1.7 Timed Automata with ϵ-Transitions

In this section we introduce into timed automata the possibility of firing non-observable transitions. The non-observable symbol is denoted, as usual, by ϵ. The study of ϵ-transitions in the timed setting has been carried out in [BGP96] and [DGP97] and has been summarized in [BPDG98].

The properties of ϵ-transitions in the timed setting are different from the classical properties of ϵ-transitions. The most important difference is that their use, unlike in the untimed case, increases the expressive power of timed automata. Moreover, the fact that infinite fair derivations of the timed transition table of a timed automaton correspond to infinite action sequences is not true any more if ϵ-transitions are used. This is because a derivation could end with an infinite cycle of ϵ symbols yielding a finite timed word. For this reason the use of non-observable events imply the possibility, for a timed automaton, to accept a finite timed word.

In [BPDG98] the possibility of accepting a finite timed word is given also by a specific device that is the same as the classical acceptance condition for finite words. A timed automaton T includes also the definition of a set F of final states. A finite timed word is accepted if the correspondent finite derivation of $\mathcal{S}(\widehat{T})$ ends in a state (q, ν) such that $q \in F$.

Finite timed words are also considered in several papers that present results of the theory of timed regular languages. In particular, the classical Kleene theorem for classical finite automata [Kle56], relating automata representation of regular languages with their algebraic characterizations as regular expressions, has been considered in the timed setting. In [ACM97, ACM02] timed regular expression are defined and a timed analogous of the Kleene theorem is shown. Another approach to the same attempt can be found in [BP99, BPT03] where a different notion of timed regular expressions is defined.

In this thesis we do not use finite timed words even when we use ϵ-transitions in some transformations that we define (see, for instance, Chapter 3). Thus, we give the following definition.

Definition 1.18 (Timed automata with ϵ-transitions)
A timed automaton with ϵ-transitions is a tuple $T = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ where $R \subseteq Q$, $R \neq \emptyset$, is a set of repeated states *and $\widehat{T} = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ is a timed transition table whose structure is as in Definition 1.1 but the labels of the edges in $\mathcal{E}$ can be both $\sigma \in \Sigma$ and ϵ.*

The class of all timed automata with ϵ-transitions is denoted by $\mathcal{T}_\epsilon$.

Timed automata with ϵ-transitions in which all edges of the form $(q, \psi, \gamma, \epsilon, q')$ are such that $\gamma = \emptyset$ are called restricted *timed automata with ϵ-transitions and are considered as a special subclass denoted by $\mathcal{T}_{\mathcal{R}\epsilon}$.*

The timed transition system $\mathcal{S}(\widehat{T})$ is defined as for timed automata without ϵ-transitions (Section 1.3). The label, timed, action and state sequences of the infinite derivations of $\mathcal{S}(\widehat{T})$ are defined as in Definition 1.3. The runs and accepted language are defined as in Definitions 1.4 and 1.15.

Note that, since $\epsilon \notin \Sigma$, the action sequences do not contain the occurrences of the ϵ-transitions and the relative times. Consequently these transitions do not occur in the timed words accepted by T.

As an example consider the timed automaton with ϵ-transitions T_{even} shown in Figure 1.13. Note that whenever x is reset to zero the automaton can execute a visible transition. As x increases above 0 the automaton must let elapse exactly 2 time units and then perform a non-visible transition resetting x. Thus, every

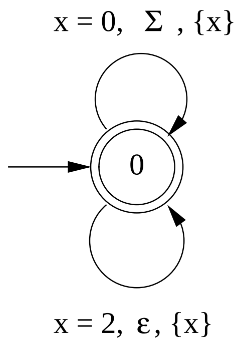

Figure 1.13: Automaton T_{even} recognizing $\mathcal{L}_{even}$

visible transition is separated from the successive by an even number of time units. Formally, the accepted language is

$$\mathcal{L}_{even} = \{(\sigma_i, t_i)_{i \geq 1} |\ \sigma_i \in \Sigma \wedge \forall i \geq 1.\ (t_i \leq t_{i+1}) \wedge (t_i = 2h,\ h \in \mathbb{N})\}$$

This particular language is used in [BPDG98] to show that

$$\mathcal{L}(\mathcal{T}) \subset \mathcal{L}(\mathcal{T}_\epsilon)$$

Indeed, timed automata are special cases of timed automata with ϵ-transitions and, in *op. cit.*, the authors show that L_{even} cannot be accepted by any timed automaton without ϵ-transitions. Thus, ϵ-transitions increase the expressive power of timed automata, unlike in the untimed classical automata.

However, in *op. cit.*, it is shown that the real increasing power is given by ϵ-transitions that reset clocks. For this reason the class $\mathcal{T}_{\mathcal{R}\epsilon}$ has been introduced. It is shown that

$$\mathcal{L}(\mathcal{T}) = \mathcal{L}(\mathcal{T}_{\mathcal{R}\epsilon})$$

Actually, the result is slightly stronger. Namely, it is shown that, for every timed automaton T in which no ϵ resetting transition lie along a cycle on the graph representing $\widehat{T}$, it is possible to construct a timed automaton without ϵ-transitions that accepts the same *divergent* timed words of T.

The fact that the result is not true for Zeno words is quite surprisingly. However, the evidence is given in [BPDG98] using the Zeno timed language $\mathcal{L}(T_z)$ accepted by the timed automaton shown in Figure 1.14. It does not contain ϵ-transitions that reset clocks and that lie along a cycle. The accepted language is

$$\mathcal{L}(T_z) = \{(a, t_i)_{i \geq 1} |\ \exists w > 0 : 0 \leq t_1 < t_2 < \ldots < t_2 + 1 - w\}$$

Every timed word of $\mathcal{L}(T_z)$ has a converging sequence of times. It is shown in *op. cit.* that this language cannot be accepted by any timed automaton without ϵ-transitions.

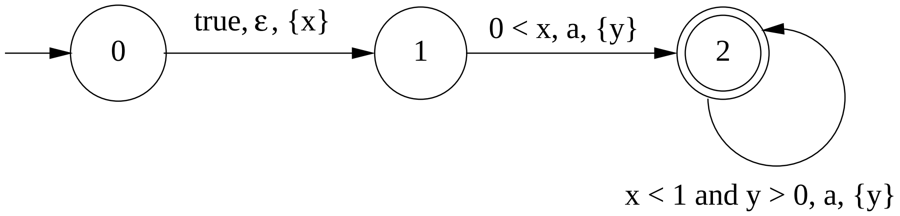

Figure 1.14: Automaton T_z

Finally, in *op. cit.*, a concept of *precise action* and a relative result are introduced, which are two useful devices in showing that a timed language cannot be accepted by any timed automaton without ϵ-transitions. We recall it here because it will be used in the proof of Proposition 3.4.

Let T be a timed automaton which uses a finite set $C \subseteq \mathbb{Q}^{\geq 0}$ of rational constants in the clock constraints. Suppose also, without loss of generality, that T is disjunction free and diagonal free (see Section 1.6.4). Let $C_{\max}$ be the maximum value of the set C and let $\delta > 0$ be such that $C \subseteq \delta\mathbb{Z}^{\geq 0}$ (δ can be chosen such that δ^{-1} is the common denominator of the numbers in C).

Consider an infinite path π, along the graph representing the automaton T, starting from an initial state q_0: $(q_0, \psi_0, \gamma_0, \sigma_0, q_1)(q_1, \psi_1, \gamma_1, \sigma_1, q_2) \cdots$

Definition 1.19 (Precise Action) *Let T and π be as above. By $TS(\pi)$ we denote the set of all possible time sequences $t_0t_1t_2\cdots$ such that $0 = t_0 \leq t_1 \leq t_2 \leq \cdots$ and there exists a run of T whose time sequence is $t_0t_1t_2\cdots$ and whose infinite path along the states of T is the same as the one of π.*

For each $n \in \mathbb{N}$, $TS_n(\pi)$ denotes the set of values $v \in \mathbb{R}^{>0}$ such that there exists a time sequence $t_0t_1t_2\cdots t_n\cdots \in TS(\pi)$ with $t_n = v$.

The occurrence of σ_n along the path π is called a precise action *in π if $TS_n(\pi)$ reduces to a singleton.*

Now we report the theorem, whose proof can be found in [BPDG98], regarding precise actions.

Theorem 1.2 *Let T, π, δ and $C_{\max}$ as above. Assume that $TS(\pi) \neq \emptyset$ and let $t \in TS_n(\pi)$.*

- *If the occurrence σ_n is a precise action in π, then $t \in \delta\mathbb{N}$. Moreover, if $t > C_{\max}$ then there exists an occurrence a_m, with $m < n$, such that a_m is also a precise action in π and the unique element of $TS_m(\pi)$ belongs to the half-open interval $[t - C_{\max}, t)$.*

- *Otherwise, $TS_n(\pi)$ is a non-empty interval (r, s) with $r \in \delta\mathbb{N}$ and $s \in \delta\mathbb{N} \cup \{\infty\}$.*

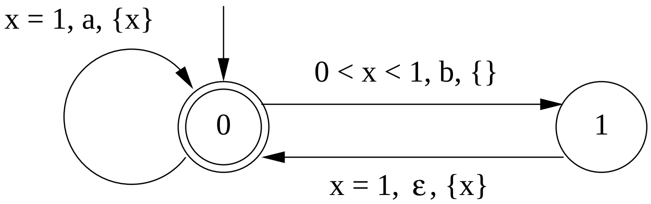

Figure 1.15: Automaton T_a

This result is a useful formal tool that can be used to show that some timed languages are not recognizable by any timed automaton (without ϵ-reset transitions). It has been used, in *op. cit.*, to show that the language accepted by the timed automaton with ϵ-transitions T_a in Figure 1.15 cannot be accepted by any automaton in $\mathcal{T}$.

The language $\mathcal{L}(T_a)$ is such that in every timed word

- in every open time interval $(i, i+1)$, where $i \geq 1$, at most a b can occur
- an a occurs at time $i+1$ if and only if a b did not occur in $(i, i+1)$.

Formally,

$$\mathcal{L}(T_a) = \{(\overline{\sigma}, \overline{t}) \mid \forall i \in \mathbb{N}.\ t_i \in (i, i+1] \wedge ((\sigma_i = a \wedge t_i = i+1) \vee (\sigma_i = b \wedge t_i \in (i, i+1)))\}$$

This example also shows that an ϵ-transition that resets clocks and that lies on a cycle cannot be removed in general.

Chapter 2

Verification of Real-time Systems

Abstract

In this chapter we introduce the task of automatic verification of real-time systems. The timed automata model introduced in Chapter 1 is used as referring formalism for modeling real-time systems and the automata-theoretic approach to verification is exposed. Important decidable problems and the fundamental region construction for timed automata are reported in detail. We then discuss implementation issues of timed automata and show that important simplifications have to be done to the theoretical model to reach implementability and practical feasibility of the verification. In particular, we introduce the formalism of timed safety automata and their use as models in the model-checking approach to the verification of properties expressed in the real-time temporal logic TCTL. The syntax and the semantics of this logic are defined with respect to the behaviors of timed safety automata. Finally, we introduce the tools KRONOS and UPPAAL that are used to perform verifications on the examples of the remaining chapters. Many concepts and notations introduced in this chapter are used, as for those of Chapter 1, throughout the remaining part of the thesis.

A practical feasible automatic verification of real-time system is one of the main objectives of the field of research in which this thesis can be placed. For this reason, after the introduction of a clear basic theoretical model in Chapter 1, we show in this chapter the main theoretical and practical formalisms and techniques that can be used to reach this objective.

We start presenting the classical railway cross example that is suitable to show the features of the timed automata formalism and to introduce some typical verification questions. In Section 2.2 we complete the exposition of the model of timed automata showing how they can be used to verify properties. To do this, we introduce the region construction and show how it can be used to solve the *emptiness problem* and the *reachability test* for timed automata. Then, we show how these two problems can be used to verify the properties of the railway cross example.

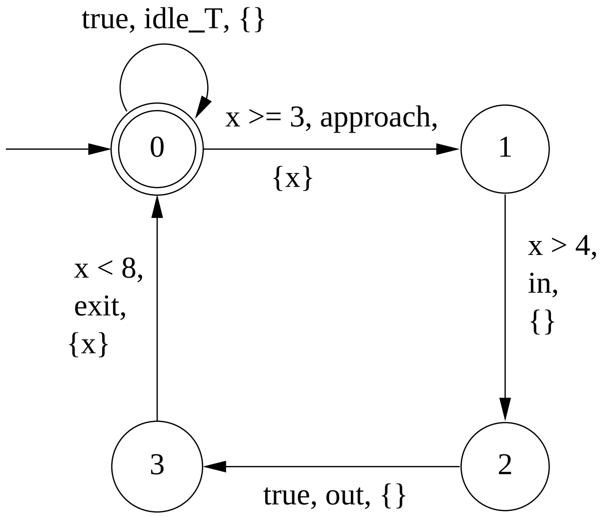

Figure 2.1: Automaton `Train`

In Section 2.3 a more practical formalism is introduced, namely the formalism of timed safety automata. Timed safety automata are a simplified model with respect to timed automata and are, in a sense, a lower level way to specify real-time systems. They are introduced in detail and their features are compared with those of timed automata. The semantic domain of timed safety automata is suitable to define the semantics of the branching-time timed temporal logic TCTL that is introduced in Section 2.3.5.

Finally, in Section 2.4, we introduce two existing automatic tools that perform model-checking of (variants of) timed safety automata.

2.1 The train example

In this section we introduce a classical example to show how a real-time system can be modeled as a parallel composition of timed automata. This example will be used also as a basis to show the features of timed automata with non-instantaneous actions that are presented in Chapter 3.

We consider an automatic controller of a gate in a railway cross. This example was introduced in [LS85] and used in [AD94]. The system is simple but useful to show basic features of the formalism and to formulate typical requirements. The real-time system is modeled by three components: the `Train`, the `Gate` and the `Controller`.

For simplicity we assume that one unit of time corresponds to a minute. In Figure 2.1 it is shown the timed automaton which models the behavior of trains. The set of symbols is {`idle_T, approach, exit, in, out`}. The automaton starts in state 0. It is not required that a train enters the railroad crossing. This is modeled

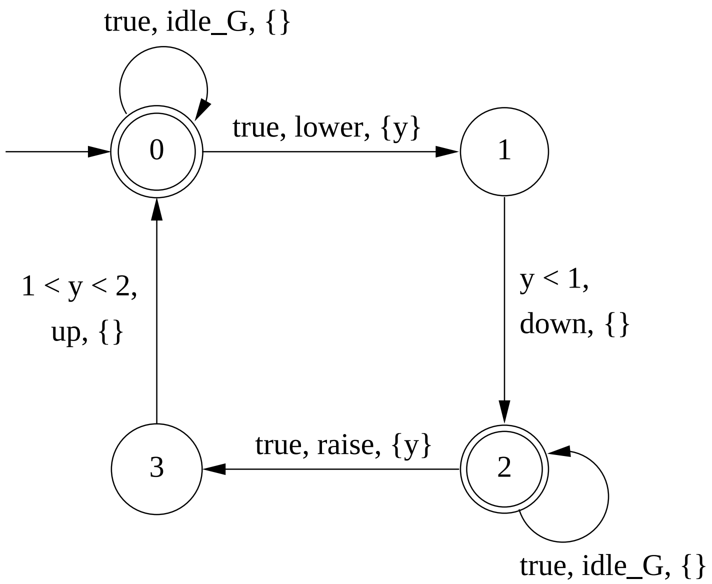

Figure 2.2: Automaton `Gate`

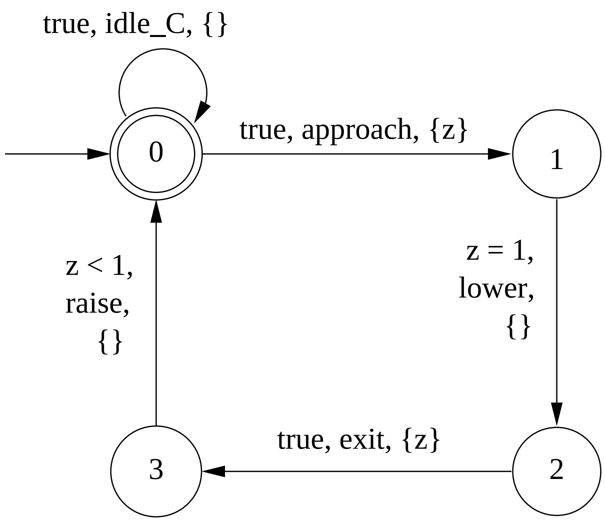

Figure 2.3: Automaton `Controller`

by the idle action which can iterate on the repeated state 0. However, a train could enter the railroad crossing any number of times, provided that at least 3 minutes elapse between the `exit` of a train and the `approach` of the subsequent (this is modeled by the reset of the clock x before entering state 0 and by the constraint $x \geq 3$ on the edge labeled `approach`). When a train is going to arrive to the railway cross it is required to communicate with the controller, by the signal `approach`, at least 4 minutes before it enters the crossing (the `in` event). The end of the crossing is indicated by the event `out` and it is not constrained, but the train has to exit the railroad crossing (`exit` event) at most 8 minutes after the `approach` signal.

The automaton modeling the gate is shown in Figure 2.2. The set of symbols is { `idle_G, lower, down, raise, up` }. In state 0 the gate is open and in state 2 it is closed. The gate can cycle on its idle action `idle_G` on state 0 or state 2 forever. This means that it can remain open or closed forever. When it is open and receives the signal `lower` from the controller it has to close (`down` action) within 1 minute. When it is closed, it can receives the signal `raise` from the controller and it must open (`up` action) within 1 to 2 minutes.

The controller is shown in Figure 2.3. It manages the signals transmission. It, as the other components, can idle forever. When it receives `approach` from a train, it sends `lower` to the gate exactly after 1 minute. When it receives `exit` from a train, it sends `raise` to the gate within 1 minute.

Consider, now, the whole system obtained by the parallel composition `Train` || `Gate` || `Controller`. The correctness requirements for this system are the followings:

1. *Safety.* Whenever the train is inside the gate, the gate must be closed.
2. *Liveness.* The gate never remains closed for more than 11 minutes.

2.2 The automata-theoretic approach to verification

The approach to verification of untimed systems using automata theory follows the following lines.

A reactive system can be modeled as a parallel composition of Büchi ω-automata S. The language accepted by S, $\mathcal{L}(S)$, represents the set of all behaviors of the modeled system. The requirements of the system can be also modeled using a Büchi automaton P that, generally, has a simpler structure than S and accepts the set of all wanted “good” behaviors of the modeled system. The verification problem is then posed as a language inclusion problem:

$$S \models P \text{ iff } \mathcal{L}(S) \subseteq \mathcal{L}(P)$$

Unfortunately, in the timed case this kind of approach cannot be used in general because it is shown in [AD94] that the timed language inclusion problem is undecidable for timed automata.

The undecidability result is due to the fact that the classes $\mathcal{T}$ and $\mathcal{M}$ are not closed under complement, i.e. given a timed automaton T over an alphabet Σ in one of these classes, there not exists in general another automaton, in the same class, that accepts a timed language which is the complement of $\mathcal{L}(T)$ with respect to the universe $\{(\Sigma^\omega, \bar{t}) \mid \forall i \in \mathbb{N}.\ t_i \leq t_{i+1}\}$.

The undecidability of language inclusion implies the undecidability of the trace equivalence between timed automata (see Definition 1.16) that can be expressed as $T_1 \approx T_2$ iff $\mathcal{L}(T_1) \subseteq \mathcal{L}(T_2) \wedge \mathcal{L}(T_2) \subseteq \mathcal{L}(T_1)$.

However, there is a subclass of timed automata that are closed under complement, namely deterministic timed Muller automata introduced in Section 1.4.3. In this case it is possible to apply the approach of verification by language inclusion. In [AD94] a procedure is outlined to check a timed language inclusion $\mathcal{L}(S) \subseteq \mathcal{L}(P)$ where S is a parallel composition of timed automata and P is a deterministic timed Muller automaton.

Another subclass of timed automata for which the language inclusion is decidable is the class of *Event-clock* automata, introduced in [AFH99].

Decidable problems for timed automata that can be used to perform an automatic verification of properties are the *reachability test* and the *emptiness problem.* They are based on the region construction which is introduced in the following section. In Section 2.2.2 it is shown how to perform verification using these tools.

2.2.1 The region construction

The fundamental idea behind the success of timed automata is the idea of clock regions. It was introduced in [AD90] and it consists in observing that, given a finite set of clock variables $\mathcal{X}$ and a finite set $\mathcal{C}$ of clock constraints on $\mathcal{X}$, i.e. $\mathcal{C} \subseteq \Psi_{\mathcal{X}}$, the infinite space of all possible clock valuations $\mathcal{V}_{\mathcal{X}}$ can be partitioned into a *finite* set of equivalence classes such that given two clock evaluations ν and ν' belonging to the same equivalence class, for every clock constraint $\psi \in \mathcal{C}$, $\nu \models \psi$ iff $\nu' \models \psi$. In [AD94] the equivalence relation inducing this partition is said to be *stable* with respect to the set of clock constraints $\mathcal{C}$.

The equivalence classes are called *clock regions* and are defined as follows. Let $\widehat{T} = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ be a timed transition table. We are supposing that the clock constraints of $\mathcal{E}$ contains only non-negative integer constants and we have showed in Section 1.6 how to reduce to this case. Let $\mathcal{C}$ be the set of clock constraints appearing in $\mathcal{E}$ and, for each $x \in \mathcal{X}$, let c_x be the greatest integer constant to which x is compared in $\mathcal{C}$. For every real number t we denote by $\lfloor t \rfloor$ the integral part of t and by $\texttt{fract}(t)$ the fractional part of t.

Definition 2.1 (Equivalence $\sim$) *The relation $\sim$ between clock valuations of $\mathcal{V}_{\mathcal{X}}$ is defined as follows: $\nu \sim \nu'$ iff all the following conditions hold*

1. *For all $x \in \mathcal{X}$ either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or $\nu(x) > c_x \wedge \nu'(x) > c_x$*

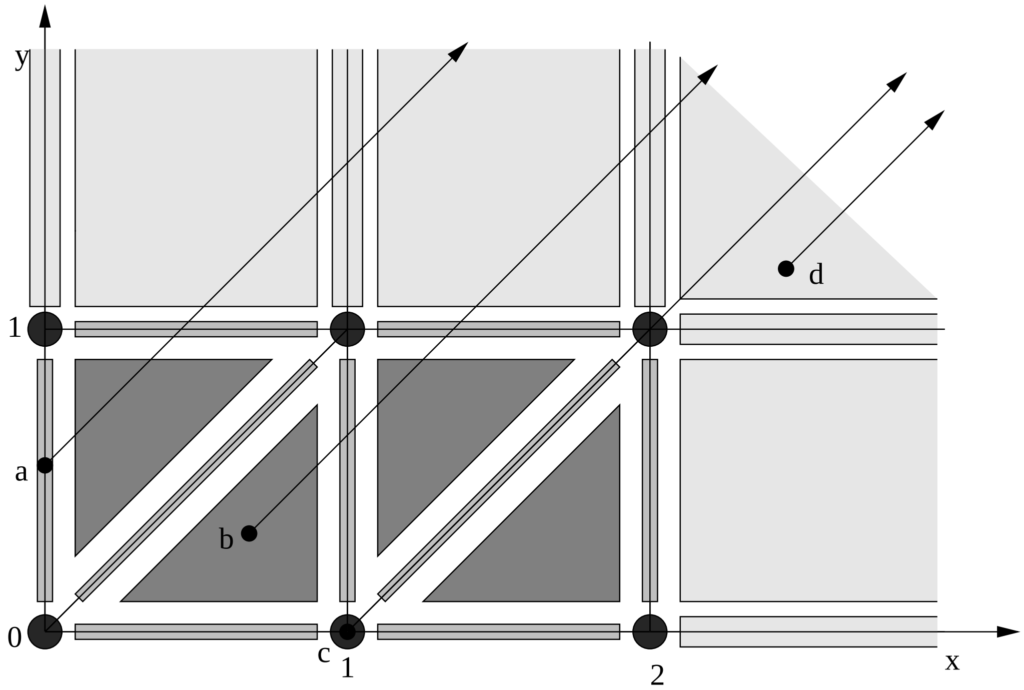

Figure 2.4: Clock regions for $\mathcal{X} = \{x, y\}$ and $c_x = 2, c_y = 1$

2. *For all* $x, y \in \mathcal{X}$ *with* $\nu(x) \leq c_x$ *and* $\nu(y) \leq c_y$, $\mathtt{fract}(\nu(x)) \leq \mathtt{fract}(\nu(y))$ *iff* $\mathtt{fract}(\nu'(x)) \leq \mathtt{fract}(\nu'(y))$

3. *For all* $x \in \mathcal{X}$ *with* $\nu(x) \leq c_x$, $\mathtt{fract}(\nu(x)) = 0$ *iff* $\mathtt{fract}(\nu'(x)) = 0$

The relation $\sim$ is an equivalence relation and the number of equivalence classes, called clock regions, that it induces is bounded by $n! \cdot 2^n \cdot \prod_{x \in \mathcal{X}}(2c_x + 2)$ where n is the number of clocks in $\mathcal{X}$. The equivalence class of a clock valuation ν is denoted by $[\nu]$ and clock regions are ranged over by $\alpha, \alpha', \ldots$ We denote by $\mathtt{Reg}(\widehat{T})$ the set of all clock regions for the timed transition table $\widehat{T}$.

A clock region $\alpha \in \mathtt{Reg}(\widehat{T})$ can be uniquely identified by specifying both

- for every clock $x \in \mathcal{X}$, one clock constraint of the set $C_x = \{x = c \mid c = 0, 1, \ldots, c_x\} \cup \{c - 1 < x < c \mid c = 1, 2, \ldots c_x\} \cup \{x > c_x\}$

- for every pair of clocks x and y, with associated constraint $c - 1 < x < c$ and $d - 1 < y < d$ (for some c and d), an inequality of the type $\mathtt{fract}(x)\#\mathtt{fract}(y)$, where $\# \in \{<, =, >\}$. Equivalently we can use a clock constraint of the form $x - y\#c$ or $y - x\#c$.

Given a clock region $\alpha \in \mathtt{Reg}(\widehat{T})$ and $x \in \mathcal{X}$ we denote by $R_T(\alpha, x)$ the unique clock constraint in C_x in the specification of α.

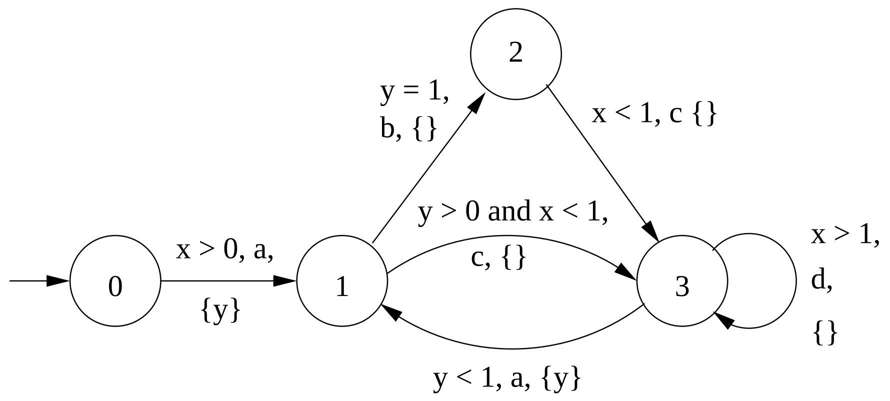

Figure 2.5: A timed transition table $\widehat{T}_r$

Given a clock region $\alpha \in \mathtt{Reg}(\widehat{T})$ and a reset $\gamma \subseteq \mathcal{X}$ we denote by $[\gamma \to 0]\alpha$ the clock region such that, for all $x \in \gamma$, the constraint in α for x is substituted by $x = 0$.

In Figure 2.4 it is shown the set of clock regions for a timed transition table with $\mathcal{X} = \{x, y\}$, $c_x = 2$ and $c_y = 1$. There are 28 regions. Note that regions can be composed only by one point, e.g. the region $[x = 1 \wedge y = 0]$ (point c in figure) or by open segments as $[x = 0 \wedge 0 < y < 1]$ (point a of the figure belongs to this region). Another type of regions are open bounded areas as $[0 < x < 1 \wedge 0 < y < 1 \wedge \mathtt{fract}(x) > \mathtt{fract}(y)]$[1] (point b of the figure belongs to this) or, finally, open unbounded areas as $[x > 2 \wedge y > 1]$ (point d of the figure belongs to this).

For every clock region α we can compute the set of time successors of α, i.e. the clock regions $[\nu + \delta]$ where $\nu \in \alpha$ and $\delta \in \mathbb{R}^{>0}$. Graphically these regions can be individuated following a diagonal line (having a 45° angle) from any point in α. We denote by $\mathtt{succ}(\alpha)$ the set of all regions that are time successors of α.

For example, in Figure 2.4, starting from point a we traverse, in order, the regions, $[0 < x < 1 \wedge 0 < y < 1 \wedge y - x > 0]$, $[0 < x < 1 \wedge y = 1]$, $[0 < x < 1 \wedge y > 1]$, $[x = 1 \wedge y > 1]$, $[1 < x < 2 \wedge y > 1]$, $[x = 2 \wedge y > 1]$, $[x > 2 \wedge y > 1]$.

Note that in this case we exit immediately from the starting region. In case of the region $[0 < x < 1 \wedge 0 < y < 1 \wedge x - y > 0]$, starting from point b, we remain in the same region for a while and then we traverse other regions.

Note that the region $[x > 2 \wedge y > 1]$ has only one successor, which is itself.

Using the concept of regions we can construct an untimed automaton that mimics the untimed behavior of a timed transition table.

Definition 2.2 (Region automaton) *Let $\widehat{T} = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$ be a timed transition table. The* region automaton *of $\widehat{T}$ is denoted by $\mathtt{Unt}(\widehat{T})$ and is the automaton $\mathtt{Unt}(\widehat{T}) = (Q', \Sigma, \mathcal{E}', B', R')$ defined as follows:*

[1] Or equivalently $[0 < x < 1 \wedge 0 < y < 1 \wedge x - y > 0]$.

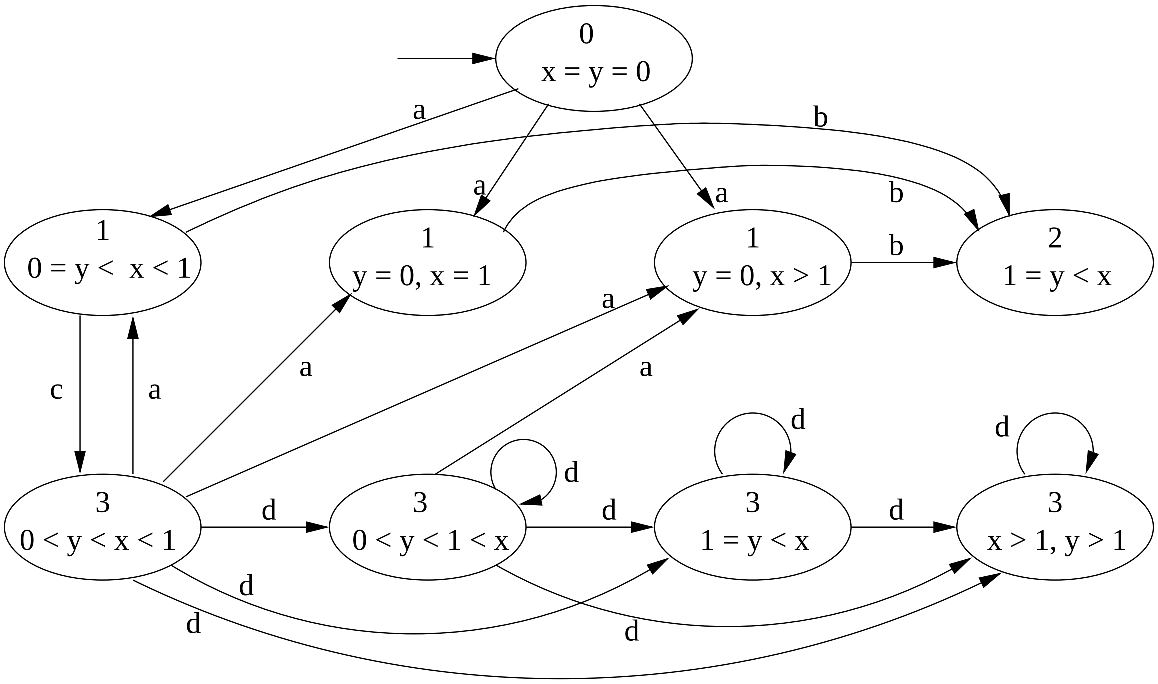

Figure 2.6: Region automaton $\texttt{Unt}(\widehat{T}_r)$

- *the states in* Q' *are pairs of* $Q \times \texttt{Reg}(\widehat{T})$*, i.e. a state is of the form* (q, α) *where* $q \in Q$ *and* $\alpha \in \texttt{Reg}(\widehat{T})$ *is a clock region*
- *the states in* B' *are of the form* $(q, [\nu_0])$ *where* $q \in B$ *and* $\nu_0(x) = 0$ *for all* $x \in \mathcal{X}$
- $((q, \alpha), \sigma, (q', [\gamma \to 0]\alpha'')) \in \mathcal{E}'$ *iff* $(q, \psi, \gamma, \sigma, q') \in \mathcal{E}$*,* $\alpha \in \texttt{Reg}(\widehat{T})$*, and* $\alpha'' \in \texttt{succ}(\alpha)$*.*

An infinite run ρ of the automaton $\texttt{Unt}(\widehat{T})$ is of the form $s_0 \xrightarrow{\sigma_0} s_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{i-1}} s_i = (q_i, \alpha_i) \xrightarrow{\sigma_i} \cdots$ where $s_0 \in B'$ and $\forall i \in \mathbb{N}. \exists (s_i, \sigma_i, s_{i+1}) \in \mathcal{E}'$. The label sequence of the run ρ is the sequence $\sigma_0\sigma_1\sigma_2\cdots$ and it is denoted by $\overline{\rho}$.

A run ρ of $\texttt{Unt}(\widehat{T})$ is *progressive* if and only if, for each clock $x \in \mathcal{X}$, there are infinitely many $i \geq 0$ such that α_i satisfies $x = 0 \vee x > c_x$. A progressive run corresponds to a divergent derivation of $\mathcal{S}(\widehat{T})$.

Let ρ be a progressive run of $\texttt{Unt}(\widehat{T})$. In [AD94] it is proved that there exists a time sequence $\overline{t}$ such that $(\overline{\rho}, \overline{t})$ is the action sequence of a fair divergent derivation of $\mathcal{S}(\widehat{T})$.

As an example, in Figure 2.5 it is shown a simple timed transition table $\widehat{T}_r$ and in Figure 2.6 is shown its region automaton.

We can now define the region Büchi automaton associated to a timed automaton.

Definition 2.3 (Region Büchi automaton)
Let $T = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton. The region Büchi automaton of T is denoted by $\mathtt{Unt}(T)$ *and is the automaton* $\mathtt{Unt}(\widehat{T})$ *together with the set of repeated states* $R' = R \times \mathtt{Reg}(\widehat{T})$.

The untiming construction given in [AD94] states that for every divergent timed word $(\overline{\sigma}, \overline{t})$ accepted by T, the infinite word $\overline{\sigma}$ is accepted by $\mathtt{Unt}(T)$ with the Büchi acceptance condition.

Of course the Büchi acceptance condition in the untimed case acts as in the timed one, i.e. a word is accepted if and only if its corresponding run ρ is such that there exists a state (q, α) that occurs infinitely many times in ρ and that belongs to R'. Büchi ω-automata define the class of ω-regular-languages.

It is in the light of the result on the untiming construction that in [AD94] the languages $\mathcal{L}(\mathcal{T})$ are called timed *regular* languages. The fact that untiming operation preserves regularity of languages is not an isolate case. A study that treats more generally the untiming construction can be found in [B9́5].

Finally, we want to remark that the region construction given in this section can be done, in the same way, if we use timed automata with ϵ transitions (see Section 1.7). This means that the decidable tests exposed in the following section can be applied also to the class $\mathcal{T}_\epsilon$.

2.2.2 Decidability Results for Verification

We can use the region Büchi automaton of a timed automaton as a structure on which we perform verification because its finiteness allows the application of several algorithms.

One important decision problem that can be solved for timed automata is the **emptiness problem**, i.e. deciding, given a timed automaton, if it accepts at least a divergent timed word. In [AD94] it is shown that this problem is PSPACE-complete. The suggested algorithm, given a timed automaton $T = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$, acts as follows:

1. Construct the region Büchi automaton $\mathtt{Unt}(T) = (Q', \Sigma, \mathcal{E}', B', R', \mathcal{X})$
2. Check if there exists, in the graph representing $\mathtt{Unt}(T)$, a cycle C such that:
   - C is reachable by a path along the graph that starts at a state in B'
   - at least one state of R' belongs to C
   - at least one state of $\{(q, \alpha) \in Q' \mid \forall x \in \mathcal{X}.\ \alpha \models (x = 0) \vee \alpha \models (x > c_x)\}$ belongs to C
3. If such a cycle C does not exist then the timed automaton T accepts an empty set of divergent timed words.

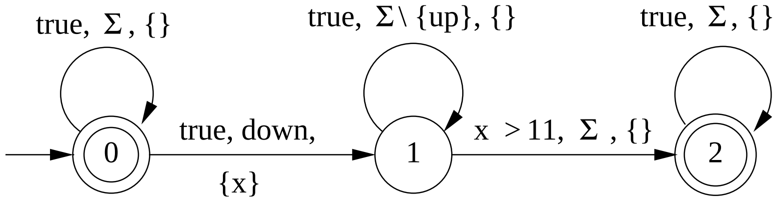

Figure 2.7: Liveness property complement

The emptiness checking can be used to verify a property of a real-time system modeled by a timed automaton T in the following way:

1. Model the complement of the property *prop* by a timed automaton $\overline{T_{prop}}$
2. Construct the parallel composition $T \parallel \overline{T_{prop}}$
3. Check if $\mathcal{L}(T \parallel \overline{T_{prop}})$ is empty using the procedure given above
4. If it is empty then T satisfies *prop*, else the property is not satisfied. In the latter case the traces of the automaton can be considered counterexamples that give diagnostic information about which situations make the property to fail.

Recall the example of Section 2.1 and consider the liveness property 2. A timed automaton modeling the complement of the property is shown in Figure 2.7. It requires that exists at least a case in which the action `close` is performed and for more than 11 subsequent time units the action `up` is not performed. Putting this automaton in parallel with the system `Train` $\parallel$ `Gate` $\parallel$ `Controller`, if the resulting automaton accepts an empty set of divergent timed words, then the property 2 is satisfied.

Another important verification that can be performed using the region Büchi automaton is the **reachability test**, i.e. deciding if a state of a timed automaton T can be reached in a divergent run of T.

To do this, given a timed automaton $T = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ and a state $q_r \in Q$ we can act as follows:

1. Construct the region Büchi automaton $\mathtt{Unt}(T) = (Q', \Sigma, \mathcal{E}', B', R', \mathcal{X})$
2. Check if there exists, in the graph representing $\mathtt{Unt}(T)$, a cycle C such that:
   - C is reachable by a path along the graph that starts at a state in B'
   - at least one state of R' belongs to C
   - at least one state of $\{(q, \alpha) \in Q' \mid \forall x \in \mathcal{X}.\ \alpha \models (x = 0) \lor \alpha \models (x > c_x)\}$ belongs to C
   - q_r belongs to C

3. If such a cycle C exists then the state q_r is reachable, else it is not reachable.

The safety property 1 of the example of Section 2.1 can be checked using reachability: we can check whether all states in which the train is inside the gate (state 2) and the gate is *not* closed (states $\{0, 1, 3\}$) cannot be reached in the parallel composition `Train` $\|$ `Gate` $\|$ `Controller`.

2.2.3 Techniques for improving efficiency

The effective implementation of the algorithms outlined in the previous section has to face the problem of the huge state explosion due to two principal factors:

- Composing in parallel different timed automata determines an exponential number of states on the parallel composition automaton that has to simulate the interleaving and the synchronization of all the actions of the components.

- The number of regions is bounded by $n! \cdot 2^n \cdot \prod_{x \in \mathcal{X}} (2c_x + 2)$ where n is the number of clocks of the given automaton.

Since the introduction of the idea of regions, a lot of researchers have studied different ways to contain the size of the graph representing the state space to explore when checking properties. Exposing the different techniques that have been developed in this field goes beyond the scope of this thesis. However, we want to briefly show a simple idea that has been fruitful, namely the idea of *clock zones*. Another important technique to increase efficiency is the use of symbolic model checking that will be treated in the following section.

Recall the equivalence classes $\mathtt{Reg}(\widehat{T})$ defined in Section 2.2.1. We have seen that the equivalence $\sim$ is *stable* with respect to the clock constraints of $\widehat{T}$. However, there exist *stable* equivalence relations with respect to the clock constraints of a given $\widehat{T}$ that correspond to a smaller number of equivalence classes than that of regions.

In [ACH$^+$92] clock zones are introduced. A clock zone is a convex union of clock regions and, like these ones, can be expressed by a conjunction of clock constraints. The most important fact is that there exists a minimization algorithm, originally introduced in *op. cit.*, that constructs a minimal zone automaton, i.e. an automaton which satisfies the same properties of the region automaton but that has a minimum number of states. The use of zones and of other suitable data structures (e.g. difference-bound matrices) have been very useful in making practically feasible the verification of properties of timed system specified by timed automata (see, for instance, [Yov96, ABG98]). As an example of how much the use of zones instead of regions can reduce the state space we have reported, in Figure 2.8, the zone automaton for the timed transition table $\widehat{T_r}$ of Figure 2.5 whose region automaton was presented in Figure 2.6.

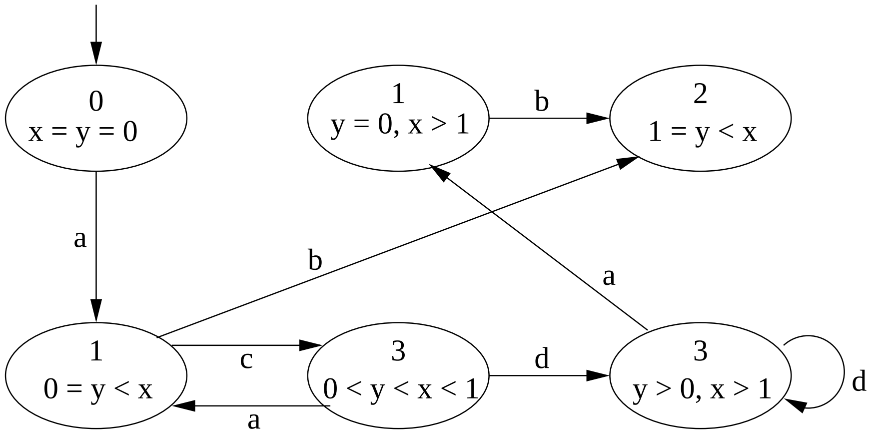

Figure 2.8: Zone automaton for T_r

2.3 The model-checking approach to verification

We have used in this thesis timed Büchi automata as the main device for specifying real-time systems. This model allows to specify typical requirements of real-time systems and the theory developed in [AD94] and in the other papers concerning the theory of timed regular languages is a good theoretical basis for the framework of automatic verification of real-time systems. In Chapter 1 we showed how a timed automaton uses fairness conditions (Definition 1.4 and the Büchi acceptance condition) to specify those derivations of the timed transition system associated to its timed transition table that represent the actual intended behaviors of the modeled system. This type of model always allows time to advance in a state (recall rule T1 of Figure 1.1 in Section 1.3). If, in a derivation, the time advances too much so that the automaton cannot perform transitions any more, then the derivation is discarded by the fairness condition of Definition 1.4 and it is not considered as a behavior of the system. Moreover, the Büchi acceptance condition may discard some other derivations that do not satisfy a condition depending on information about the infinite sequence of states of the derivation ($\mathtt{inf}(r)$).

It is easy to realize that an interpreter that attempts to generate a run of a given timed automaton T cannot decide, being in one state of $\mathcal{S}(\widehat{T})$ at a certain point of a derivation, how to progress into a run of T using only the information of the state. This is because the required information to do so is infinitary and global to the automaton.

For this reason, in verification and model checking tools, the model used is different. The information required for the progress at each state is *local*: the states of T are equipped with an invariant condition on the values of the clocks which must be always true when the control is in the state. This forces the automaton to perform a transition to leave the state when the condition is going to become false. This feature can be used to assure the progress of every derivation of the timed transition system defining the semantics of the automaton as the set of all infinite derivations of the timed transition system even if they end with an infinite sequence of δ-transitions (this is possible only if the invariant condition is *true*). Acceptance conditions are not used and the implementation of the model is relatively easy. This model was introduced in [HNSY94] and is used in all simulators and verification tools developed for timed automata. In *op. cit.* the focus is on divergence-safe real-time systems, a subclass of the real-time systems that can be specified by timed automata. We define this class in the following using Timed Safety Automata, introduced in *op. cit.*, which are representations of divergence-safe real-time systems.

A practical difference between the two models is in the process of specification and/or design of a real-time system. It is in that process that one decides the progress model to adopt and then, accordingly, writes the automaton. Timed automata are in a sense a higher level formalism with respect to timed safety automata and often a real time system modeled by a timed automaton can be rewritten, adding details and making some approximations, into a timed safety automaton. The result cannot be in general an equivalent system because, to reach implementability, timed safety automata forbid *unbounded inevitability*. We will clarify this aspect by an example below.

Moreover, timed safety automata have a state-base semantics. The model of real-time systems that they use is the so-called *branching-time* model in which the behaviors of a system are sequences of states where each state is typically represented by the values of a set of boolean variables. This type of semantics is useful for defining the derivations of timed safety automata as models of formulas of timed branching-time temporal logics. We introduce in Section 2.3.5 the logic TCTL (Timed Computational Tree Logic) introduced in [ACD93]. It is a real-time extension of CTL [CES86]. The model checking of this logic with respect to timed safety automata is implemented in the tool KRONOS (see Section 2.4.1) which is used in Chapter 3. Moreover, timed safety automata with some additional features can be model checked in UPPAAL (see Section 2.4.2) with respect to formulas that use a restricted version of the branching time operators of TCTL. UPPAAL is used in Chapters 4 and 5 to do some verifications.

It is important for us to remark that, often, new features added to timed automata can be easily transported to the model of timed safety automata to make the automatic verification effective and feasible. In this way, the extensions can be defined upon a clear theoretical basis and implementation details can be considered a separate task to face to. In this thesis this is often done. In Chapter 3 we introduce non-instantaneous actions into the model of timed automata. The translations that we give into timed automata with ϵ-transitions can be easily rephrased in translations into timed safety automata, which are used at the end of the chapter to specify the railway cross system (making some approximations). In Chapter 4 we introduce urgent transitions for timed automata but we show that it is possible to define both urgent transitions and the relative translation using timed safety automata as underlying model. Finally, in Chapter 5, we define timed non-interference for timed automata, but to reach decidability and effectiveness we give a state-base characterization of the property and use timed safety automata to perform the verification.

2.3.1 Timed Safety Automata

Timed safety automata specify real-time systems using the state-based branching-time model, i.e. as sequences of states. In this setting labels on the transitions are useless because the observable objects of the system are the states. However, labels on the edges are useful in defining the synchronized product to recognize synchronization transitions. For this reason, differently from the formulation given in [HNSY94], we put labels on the states. As a matter of fact, they are included in the tool KRONOS for this purpose and also note that they are useful to define time-abstract bisimulations in [Yov96, TY01].

The main purpose in the definition of timed safety automata is the implementability and, hence, the efficiency. To achieve these objectives, they are defined in a way that minimizes the possible redundancies and that simplifies as much as possible the task of verification. Recall Section 1.6 in which several simplifications on the syntax of timed automata, which can be used without loss of generality, are introduced. In particular, for timed safety automata we use the following grammar for the clock constraints

$$\begin{array}{rl}\psi ::= & true \\ & \mid x\#c \\ & \mid x-y\#c \\ & \mid \psi \wedge \psi\end{array}$$

where x, y are clocks, $\# \in \{<, \leq, >, \geq\}$ and $c \in \mathbb{N}$. Note that the diagonal constraints are retained because their elimination introduces a consistent number of states and this is not good from the point of view of efficiency.

The invariant conditions of the states are expressed as clock constraints, but they have additional restrictions.

Definition 2.4 (Past-closed clock constraints) *Let $\psi \in \Psi_{\mathcal{X}}$ be a clock constraint. ψ is* past-closed *iff for each clock valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ and for each $\delta \in \mathbb{R}^{>0}$, it holds*

$$\nu + \delta \models \psi \Rightarrow \nu \models \psi$$

Examples of past closed clock constraints are $\psi = true$ or constraints that impose upper bounds on the values of clock, e.g. $\psi = x < 4 \wedge y \leq 3$.

Definition 2.5 (Timed safety automata) *A timed safety automaton A is a tuple $(Q, \Sigma, \mathcal{E}, \mathrm{Inv}, \mathcal{X})$ where Q is a finite set of states or locations, Σ is a finite alphabet of symbols and $\mathcal{X}$ is a finite set of clock variables.*

A1	$\dfrac{\delta \in \mathbb{R}^{>0},\ \forall \delta' \in \mathbb{R}^{>0}.\ \delta' \leq \delta \Rightarrow \nu + \delta' \models \mathrm{Inv}(q)}{(q, \nu) \xrightarrow{\delta} (q, \nu + \delta)}$
A2	$\dfrac{(q, \psi, \gamma, \sigma, q') \in \mathcal{E},\ \nu \models \psi,\ \nu \backslash \gamma \models \mathrm{Inv}(q')}{(q, \nu) \xrightarrow{\sigma} (q', \nu \backslash \gamma)}$

Figure 2.9: Rules for the transition relation of $\mathcal{S}(A)$

Inv *is a function assigning to every state its invariant:* $\mathrm{Inv}\colon Q \mapsto \Psi_{\mathcal{X}}$ *and it is such that for all* $q \in Q$ $\mathrm{Inv}(q)$ *is a past-closed constraint.*

$\mathcal{E} \subseteq (Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times Q)$ *is a finite set of edges. If* $e = (q, \psi, \gamma, \sigma, q')$ *is an edge, then* q *is the source,* ψ *is the clock constraint,* $\gamma \subseteq \mathcal{X}$ *is the reset set,* σ *is the label and* q' *is the target.*

The class of all timed safety automata is denoted by $\mathcal{A}$ *and timed safety automata are ranged over by* $A, A', \ldots, A_1, A_2, \ldots$

Note that timed safety automata are exactly timed transition tables without the set of initial states plus the invariant assigning function. As for timed transition tables we associate to a timed safety automaton A a timed transition system $\mathcal{S}(A)$ whose structure is exactly the same but the rules are modified to take into account invariants. They are showed in Figure 2.9.

Note that Rule A1 can let a time δ elapse only if this respects the state invariant, i.e. if the clock valuation continuously satisfies the state invariant while the time increases by δ time units. Moreover, a transition can be executed only if its clock constraint is satisfied by the current clock valuation *and* the clock valuation after the reset satisfies the invariant of the target state. This is expressed in Rule A2.

We define now the possible behaviors of a given timed safety automaton A. We define derivations that start at a given state of the timed transition system $\mathcal{S}(A)$.

Definition 2.6 ((q, ν)-paths) *Let* $A = (Q, \Sigma, \mathcal{E}, \mathrm{Inv}, \mathcal{X})$ *be a timed safety automaton and let* (q, ν) *be a state of the timed transition system* $\mathcal{S}(A)$.

A (q, ν)*-path, denoted by* $\pi_{(q,\nu)}$*, is an infinite derivation of the timed transition system* $\mathcal{S}(A)$ *of the form* $s_0 = (q, \nu) \xrightarrow{l_0} s_1 \xrightarrow{l_1} s_2 \cdots$.

The set of all (q, ν)*-paths, i.e. those infinite derivations that start at* (q, ν)*, is denoted by* $\Pi_A(q, \nu)$*. The set of all paths of* A *is the set*

$$\Pi_A = \bigcup_{(q,\nu) \in (Q \times \mathcal{V}_{\mathcal{X}})} \Pi_A(q, \nu)$$

The time, label, action and state sequences for (q, ν)-paths are defined in the same way as for derivations of timed transition tables (Definition 1.3)

While the semantics of a timed automaton T is the timed language $\mathcal{L}(T)$, the semantics of a timed safety automaton A is the set of all (q, ν)-paths, Π_A. The motivation of this definition can be found below when we use timed safety automata as models of formulas of TCTL. Although the denotations are rather different objects, the underlying intuitive behaviors that are modeled by timed automata and timed safety automata are similar. Giving an initial condition (q_0, ν_0) to a timed safety automaton and considering the set of all (q_0, ν_0)-paths as derivations of a timed transition table, we can compare the two formalism and make some remarks on their features.

2.3.2 Expressiveness of timed safety automata

Figure 2.10 shows a timed safety automaton A_0. It is an attempt to simulate the behaviors of the timed transition table T_0 of Figure 1.2. Note that the definition of the (q, ν)-paths does not require any fairness condition on the derivations. In particular the fairness condition of Definition 1.4 for timed transition tables is not imposed. This means that a timed safety automaton can remain in a state and let the time advance forever if the state invariant allows it to do so (i.e. it is *true*). Thus, in state 0 of A_0, it is not required that the automaton performs any transition. This is an important difference between timed automata and timed safety automata, namely timed safety automata cannot express *unbounded inevitability* (in branching time temporal logics it is often expressed by the notation $\forall\Diamond$). If an event/action (represented by a transition between states) has to occur in the system, then it must occur within a predefined amount of time. A behavior in which this time is not specified, but it is known that the event/action will eventually occur, cannot be modeled by timed safety automata. Timed automata can do this because fair derivations are considered.

The main motivation for this limitation of the model of timed safety automata is a practical one. In [HNSY94] the authors develop a symbolic approach to model checking in dense real-time and, to do this, they want to express the algorithm as a fix-point computation on suitable operators. This requires the definition of a next-state operator that has to be used during the fix-point computation. The denseness of the time domain force the time to advance, in a next-state relation, by an infinitesimal amount but, on the other hand, the time is required to diverge. A proper next-state relation can be defined for timed safety automata because the time that the system can spend in a state is restricted by the state invariants, i.e. by upper bounds, and thus the next-state relation is not required to let the time diverge. Such a relation allows to compute reachability or possibility ($\exists\Diamond$) and its dual, invariance ($\forall\Box$). Moreover, since timed safety automata cannot express unbounded inevitability, checking such a requirement reduces to checking bounded

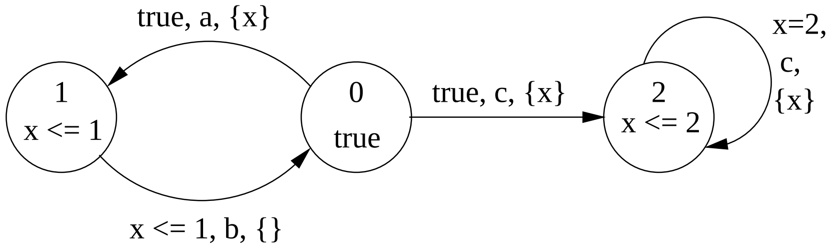

Figure 2.10: Timed Safety Automaton A_0

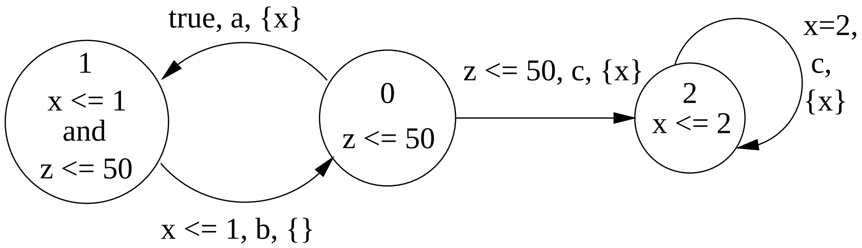

Figure 2.11: Timed Safety Automaton A_0 with additional constraints

inevitability which, in turn, can be reduced to the check of an invariance. This is because time may not progress beyond the upper bound of any event without the event occurring.

Returning to our example, if A_0 performs the transition from state 0 to state 1 (labeled a) then the real-time requirement that the transition labeled b is performed within one time unit can be expressed simply resetting the clock x when the state 1 is entered and setting the state invariant of state 1 to $x \leq 1$. In every derivation with the rules of Figure 2.9 each occurrence of a is followed by a b within 1 time unit. In state 2 of the automaton A_0 it is used the same technique to impose a behavior in which the transition labeled c is taken infinitely many times exactly every 2 time units.

The timed transition table T_0 is used in Section 1.4.1 to define a timed automaton giving the set of repeated states $\{2\}$. We observed that the Büchi acceptance condition is a further fairness condition on the derivations and that this set of repeated states expresses the requirement that, in every run of the timed automaton, *eventually* a transition labeled c is taken and a cycle of c's ends every timed word accepted by the automaton. This, again, is an unbounded inevitability requirement and it cannot be modeled by a timed safety automaton.

The best that we can do to approximate the behavior of the timed automaton T_0 is to impose that the first c is taken within a certain time, say 50 time units. This can be done adding a new clock variable z to A_0 and putting the invariant condition $z \leq 50$ both on state 0 and state 1. Although it is not needed, to underline the required behavior we can add the constraint $z \leq 50$ to the transition from state 0 to state 2 (Figure 2.11).

2.3.3 Parallel composition

The first aim of the definition of timed safety automata is implementability. Thus, they must be a proper formalism to model real life systems and to effectively perform automatic verification. For these reasons a parallel composition operator used to do modular specification has to be defined.

The parallel composition between two timed safety automata is very similar to the synchronized product between timed transition tables of Definition 1.7. In addiction to this the definition has to specify the invariants of the states of the parallel composition.

Given two timed safety automata A_1 and A_2 having a disjoint set of clocks their parallel composition is a timed safety automaton denoted by $A_1 \mid A_2$. The set of states, the edges, the alphabet and the set of clocks of the parallel composition are defined as in Definition 1.7. The invariance function $\mathrm{Inv}_{A_1|A_2}$ is such that

$$\forall (q_1, q_2) \in Q_1 \times Q_2.\ \mathrm{Inv}_{A_1|A_2}(q_1, q_2) = \mathrm{Inv}_{A_1}(q_1) \wedge \mathrm{Inv}_{A_2}(q_2)$$

where Q_1 (Inv_{A_1}) and Q_2 (Inv_{A_2}) are the set of states (invariants assigning functions) of A_1 and A_2 respectively.

2.3.4 Divergence-safe real-time systems

Another aspect that we have to consider is the non-divergent (Zeno) (q, ν)-paths. We have seen in Chapter 1 that Zeno timed words have several properties and characteristics that have to be considered in the theory of timed languages. However, when applying the theory to specification and verification of real-time systems, they are not meaningful and have to be ruled out. The first aim of the use of timed safety automata is to model *implementable* real-time systems and thus, in this setting, Zeno derivations are not considered. In [HNSY94] this aspect is treated giving a fix-point algorithm (that has been implemented in KRONOS) that, given a timed safety automaton A, individuates all the states of $\mathcal{S}(A)$ from which the time cannot diverge.

Definition 2.7 (Zeno states) *Let A be a timed safety automaton and let $\mathcal{S}(A)$ be its associated timed transition system. A state (q, ν) of $\mathcal{S}(A)$ is a* Zeno state *if and only if given any (q, ν)-paths, its time sequence $\bar{t}$ is convergent, i.e. $\exists \gamma \in \mathbb{R}^{>0}$: $\forall i \in \mathbb{N}$. $t_i < \gamma$.*

A state (q, ν) of $\mathcal{S}(A)$ is nonZeno *if and only if there exists a (q, ν)-path whose time sequence $\bar{t}$ is divergent, i.e. $\forall \tau \in \mathbb{R}^{>0}$. $\exists i \in \mathbb{N}$: $t_i > \tau$.*

Only nonZeno states are allowed in the timed transition system of timed safety automata. For instance, consider the automaton in Figure 2.12. The state of the form (q, ν) with $q \in \{0, 1\}$ and $4 < \nu(x) \leq 5$ are Zeno states. They can be detected automatically using the algorithm given in [HNSY94] and can be easily eliminated.

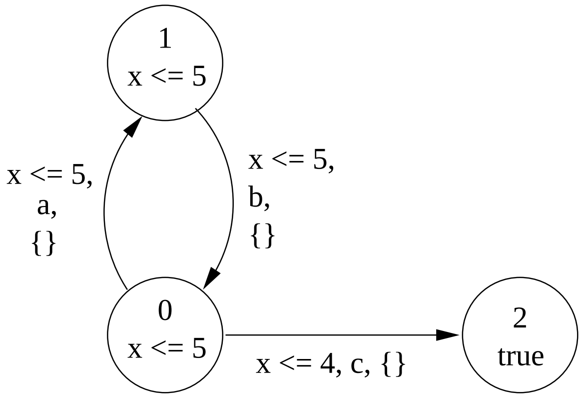

Figure 2.12: A Timed Safety Automaton with Zeno states

For instance, in the automaton in Figure 2.12 a solution is to turn the constraint $x \leq 4$ in $x \leq 5$. Another solution is to turn the invariants and constraints of states 0 and 1 into $x \leq 4$.

Finally, note that a nonZeno state is such that *there exists at least one* divergent path from it. Obviously, as for timed automata, the transition system can always generate convergent derivations, but only divergent runs from a state are considered behaviors of the modeled system. Thus, if all the states of a timed safety automaton are nonZeno then there exists always a divergent path starting from each of them. This is what is needed in the following to define computational trees denoted by a TCTL formula.

Definition 2.8 (Divergent paths) *Let A be a timed safety automaton and let (q, ν) be a state of its associated timed transition system $\mathcal{S}(A)$.*

The set $\Pi_A^\infty(q, \nu)$ is the set of all (q, ν)-paths whose time sequence is divergent. The set of all divergent paths of A is

$$\Pi_A^\infty = \bigcup_{(q,\nu)\in(Q\times\mathcal{V}_\mathcal{X})} \Pi_A^\infty(q, \nu)$$

The set Π_A^∞ has to be considered the intended semantics of the timed safety automaton A. This set of derivations, that can be seen as a set of sequences of $\mathcal{S}(A)$ states[2], satisfies the properties of fusion, suffix and stutter closure [HNSY94]. Moreover, the property of safe-divergence, which is an adapted version of the safety in linear time [ADS86], is satisfied. Namely, for every *divergent* derivation $\pi_{(q,\nu)}$, if all finite prefixes of $\pi_{(q,\nu)}$ are prefixes of derivations in Π_A^∞, then $\pi_{(q,\nu)} \in \Pi_A^\infty$.

All these properties are needed to define the semantics of a TCTL formula. The fundamental observation is that, given a nonZeno state (q, ν), the set of (q, ν)-paths can be represented as a tree whose nodes are states of $\mathcal{S}(A)$. The root is the state (q, ν) and the edges of the tree represent all possible transitions.

[2]Here we refer to every state s_i of a (q, ν)-path, which is different from the state sequence.

2.3.5 The real-time logic TCTL

In this section we introduce the real-time temporal logic TCTL (Timed Computational Tree Logic). It is an extension of the Computational Tree Logic (CTL) of [CES86] and was introduced in [ACD93] in a weaker version of that we present here[3]. A model checking algorithm based on the region graph of a given timed graph (a timed automaton without labels and without acceptance conditions) and on additional fairness checks was given in [ACD93]. Moreover, in *op. cit.*, the authors show that the model-checking problem for TCTL is PSPACE-complete.

In [HNSY94] a symbolic approach to model checking has been introduced for a timed version of the μ-calculus, the timed μ-calculus T_μ. It is shown that the approach can be used also for the model checking of TCTL formulas on divergence-safe real-time systems represented by timed safety automata. The tool KRONOS implements this approach and also the tool UPPAAL uses a restricted version of this logic as language for requirements. The definition of the syntax and of the semantics given in this section is close to the approach of [Yov96].

Let $A = (Q, \Sigma, \mathcal{E}, \mathrm{Inv}, \mathcal{X})$ be a timed safety automaton and let $\mathcal{Z}$ be a set of clock variables disjoint from $\mathcal{X}$. Moreover, let P be a set of boolean variables used to describe observable information on the states of A and let $\mathcal{P}\colon Q \to \wp(P)$ a function assigning to every state q in Q the set of boolean variable of P that are true in q. The set of formulas $\Phi_{\mathcal{X},\mathcal{Z},\mathcal{P}}$ is defined by the following grammar:

$$
\begin{array}{rcl}
\varphi & ::= & \psi \\
& & \mid b \\
& & \mid z.\varphi \\
& & \mid \neg\varphi \\
& & \mid \varphi_1 \vee \varphi_2 \\
& & \mid \varphi_1 \,\exists\mathcal{U}\, \varphi_2 \\
& & \mid \varphi_1 \,\forall\mathcal{U}\, \varphi_2
\end{array}
$$

where $\psi \in \Psi_{\mathcal{X}\cup\mathcal{Z}}$ is a clock constraint in which also clocks of $\mathcal{Z}$ can occur, $b \in P$ is a boolean variable and z is a clock variable in $\mathcal{Z}$. Clock variables of the set $\mathcal{Z}$ can occur free or bounded by a freeze clock quantifier "$z.$". The set of clock variables that occur free in a formula φ is defined as follows, where we indicate by $\texttt{clk\_var}(\psi)$ the set of clock variables occurring in a clock constraint ψ:

$$
\begin{array}{ll}
\texttt{free}(\psi) & = \texttt{clk\_var}(\psi) \\
\texttt{free}(b) & = \emptyset \\
\texttt{free}(z.\varphi) & = \texttt{free}(\varphi) - \{z\} \\
\texttt{free}(\neg\varphi) & = \texttt{free}(\varphi) \\
\texttt{free}(\varphi_1 \vee \varphi_2) & = \texttt{free}(\varphi_1) \cup \texttt{free}(\varphi_2) \\
\texttt{free}(\varphi_1 \,\exists\mathcal{U}\, \varphi_2) & = \texttt{free}(\varphi_1) \cup \texttt{free}(\varphi_2) \\
\texttt{free}(\varphi_1 \,\forall\mathcal{U}\, \varphi_2) & = \texttt{free}(\varphi_1) \cup \texttt{free}(\varphi_2)
\end{array}
$$

[3] In particular, the freeze clock quantifier $z.\varphi$ was not included.

A TCTL formula φ is *closed* if and only if $\mathtt{free}(\varphi) \subseteq \mathcal{X}$ i.e. if every occurrence of a clock variable that does not belong to the timed safety automaton is bounded by a freeze quantifier.

The truth value of a closed TCTL formula $\varphi \in \Phi_{\mathcal{X},\mathcal{Z},\mathcal{P}}$ is given by a satisfaction relation $\models_{\mathrm{TCTL}}$ defined in the following by structural induction. We have seen in the previous section that the set of all divergent (q,ν)-paths, Π_A^∞, is a suitable semantic domain for a branching time logic. The relation $\models_{\mathrm{TCTL}}$ specifies if φ is true or false in an *extended* state of the transition system $\mathcal{S}(A)$. The modal operator $\mathcal{U}$ (until) ranges over the divergent (q,ν)-paths of $\Pi_A^\infty(q,\nu)$.

An *extended* state (q,ν,ζ) is such that (q,ν) is a state of $\mathcal{S}(A)$ and $\zeta \in \mathcal{V}_{\mathcal{Z}}$ is a clock valuation for the (freeze quantified) clocks $z \in \mathcal{Z}$ of φ.

Let $\pi_{(q,\nu)}$: $s_0 = (q,\nu) \xrightarrow{l_0} s_1 \xrightarrow{l_1} s_2 \cdots$ be a (q,ν)-path of $\mathcal{S}(A)$. A *position* p of $\pi_{(q,\nu)}$ is a pair $(i,\delta) \in \mathbb{N} \times \mathbb{R}^{\geq 0}$. Each position (i,δ) represents the set of states through which the derivation $\pi_{(q,\nu)}$ passes while time elapses from state s_i to state s_{i+1} in the following way. The pair $(i,0)$ represents the exact position i of the derivation $\pi_{(q,\nu)}$ and for all δ such that $0 \leq \delta < (t_{i+1} - t_i)$[4] the position (i,δ) represents the intermediate "state" between s_i and s_{i+1} of the derivation in which a time δ has elapsed. A natural order $\prec$ is defined between positions:

$$(i,\delta) \prec (j,\delta') \text{ iff } i < j \vee (i = j \wedge \delta < \delta')$$

The set of positions of $\pi_{(q,\nu)}$ is denoted by $\mathtt{Pos}(\pi_{(q,\nu)})$. For each position $p = (i,\delta) \in \mathtt{Pos}(\pi_{(q,\nu)})$ we denote by $\Delta(p)$ the time that has elapsed from position $(0,0)$ to position p:

$$\Delta(p) = \delta + \sum_{0 \leq j \leq i-1 \,\wedge\, l_j \in \mathbb{R}^{>0}} l_j$$

The following is the definition of $\models_{\mathrm{TCTL}}$:

$$
\begin{array}{lll}
(q,\nu,\zeta) \models_{\mathrm{TCTL}} \psi & \text{iff} & \nu \cup \zeta \models \psi \\
(q,\nu,\zeta) \models_{\mathrm{TCTL}} b & \text{iff} & b \in \mathcal{P}(q) \\
(q,\nu,\zeta) \models_{\mathrm{TCTL}} z.\varphi & \text{iff} & (q,\nu,\zeta\backslash\{z\}) \models_{\mathrm{TCTL}} \varphi \\
(q,\nu,\zeta) \models_{\mathrm{TCTL}} \neg\varphi & \text{iff} & (q,\nu,\zeta) \not\models_{\mathrm{TCTL}} \varphi \\
(q,\nu,\zeta) \models_{\mathrm{TCTL}} \varphi_1 \vee \varphi_2 & \text{iff} & (q,\nu,\zeta) \models_{\mathrm{TCTL}} \varphi_1 \text{ or } (q,\nu,\zeta) \models_{\mathrm{TCTL}} \varphi_2 \\
(q,\nu,\zeta) \models_{\mathrm{TCTL}} \varphi_1 \,\exists\mathcal{U}\, \varphi_2 & \text{iff} & \exists\ \pi_{(q,\nu)} \in \Pi_A^\infty(q,\nu){:}\ \exists p = (i,\delta) \in \mathtt{Pos}(\pi_{(q,\nu)}){:} \\
& & \quad s_i = (q_i,\nu_i) \wedge (q_i, \nu_i + \delta, \zeta + \Delta(p)) \models_{\mathrm{TCTL}} \varphi_2 \\
& & \quad \wedge\ \forall p' = (j,\delta') \in \mathtt{Pos}(\pi_{(q,\nu)}).\ (p' \prec p \wedge s_j = (q_j,\nu_j)) \\
& & \qquad\quad \Rightarrow (q_j, \nu_j + \delta', \zeta + \Delta(p')) \models_{\mathrm{TCTL}} \varphi_1 \vee \varphi_2 \\
(q,\nu,\zeta) \models_{\mathrm{TCTL}} \varphi_1 \,\forall\mathcal{U}\, \varphi_2 & \text{iff} & \forall\ \pi_{(q,\nu)} \in \Pi_A^\infty(q,\nu).\ \exists p = (i,\delta) \in \mathtt{Pos}(\pi_{(q,\nu)}){:} \\
& & \quad s_i = (q_i,\nu_i) \wedge (q_i, \nu_i + \delta, \zeta + \Delta(p)) \models_{\mathrm{TCTL}} \varphi_2 \\
& & \quad \wedge\ \forall p' = (j,\delta') \in \mathtt{Pos}(\pi_{(q,\nu)}).\ (p' \prec p \wedge s_j = (q_j,\nu_j)) \\
& & \qquad\quad \Rightarrow (q_j, \nu_j + \delta', \zeta + \Delta(p')) \models_{\mathrm{TCTL}} \varphi_1 \vee \varphi_2
\end{array}
$$

[4] t_i and t_{i+1} represent the times of the time sequence associated to the derivation $\pi_{(q,\nu)}$.

We say that a state (q, ν) of $\mathcal{S}(A)$ satisfies a formula φ if and only if it holds that $(q, \nu, \zeta_0) \models_{\mathrm{TCTL}} \varphi$, where ζ_0 is such that $\forall z \in \mathcal{Z}.\ \zeta_0(z) = 0$. The timed safety automaton A satisfies a formula φ if and only if every state of $\mathcal{S}(A)$ satisfies φ.

The $\mathcal{U}$ operator is the "until" operator. A state (q, ν) satisfies $\varphi_1 \exists\mathcal{U} \varphi_2$ if and only if there exists a (q, ν)-path along which a state satisfying φ_2 is reachable and all intermediate states starting from (q, ν) satisfy φ_1 or φ_2. The formula $\varphi_1 \forall\mathcal{U} \varphi_2$ requires the same, but for all (q, ν)-paths.

In the following we give some useful macro that are often used to increase the readability of a TCTL formula. For each of them we show how it can be translated into a formula written in the basic syntax given above. The usual abbreviations of the classical logic are not given but are used with their usual meaning.

$\exists\Diamond\varphi$ is the formula to express *reachability*. It is satisfied by a state (q, ν) iff there exists a (q, ν)-path in which eventually a state satisfying φ is reached. The translation is $true \exists\mathcal{U} \varphi$.

$\forall\Box\varphi$ expresses *invariance*. It is satisfied by a state (q, ν) iff φ is satisfied in all states reachable along all (q, ν)-paths. The translation, as usual, is $\neg\exists\Diamond\neg\varphi$.

$\forall\Diamond\varphi$ expresses *inevitability*. It is satisfied by a state (q, ν) iff in all (q, ν)-paths a state in which φ is satisfied is reachable. The translation is $true \forall\mathcal{U} \varphi$.

$\exists\Box\varphi$ expresses *possible invariance*. A state (q, ν) satisfies it iff there exists a (q, ν)-path along which the formula φ is satisfied in all reachable states. The translation is $\neg\forall\Diamond\neg\varphi$.

$\exists\Diamond_{\leq c}\varphi$ is *bounded reachability*. A state (q, ν) satisfies it iff there exists a (q, ν)-path along which a state satisfying φ is reachable within c time units. The translation uses the freeze quantifier: $z.\exists\Diamond(\varphi \wedge z \leq c)$.

$\forall\Diamond_{\leq c}\varphi$ is *bounded inevitability*. A state (q, ν) satisfies it iff in all (q, ν)-paths a state satisfying φ is reachable within c time units. The translation is $z.\forall\Diamond(\varphi \wedge z \leq c)$.

Using the freeze quantification a lot of other operators can be defined using different clock constraints on the frozen clock.

Usually, when specifying a property for a given timed safety automaton A, the TCTL formula has the form $\varphi_{\mathrm{init}} \to \varphi$ where φ_{init} is a formula specifying the initial conditions. Generally, such a formula is simply a conjunction of boolean variables that are true only on those states of A that we want to consider initial states in the same sense of initial states of timed automata.

Examples of specifications given as TCTL formulas can be found in all subsequent chapters when we specify the examples as timed safety automata and use a tool to perform automatic verification.

In [HNSY94] it is shown that we can check if a timed safety automaton is nonZeno verifying that the following TCTL formula is satisfied:

$$\varphi_{\text{init}} \rightarrow \forall\Box(\exists\Diamond_{=1} true) \tag{2.1}$$

The idea is the same as the one exposed in Section 1.6.2 in the timed automata setting.

The model checking problem for the TCTL logic is decidable and it is shown that it is PSPACE-complete [ACD93]. In the same paper the authors give a model-checking algorithm for a weaker TCTL on different objects than timed safety automata, namely timed graphs. The algorithm is based on the region construction (see Section 2.2.1). In [HNSY94] a symbolic model-checking algorithm is given for TCTL and for a timed version of the μ-calculus. It can be properly defined only if the timed safety automaton to model-check is nonZeno. For this reason the authors give a fixpoint algorithm that can be used to check the formula 2.1 on any timed safety automaton. If some of its states result to be not nonZeno they can be modified in order to eliminate the Zeno derivations as we saw in Section 2.3.4. A good survey on the algorithmic techniques and data structures introduced to perform an efficient model-checking of timed safety automata can be found in [Yov96].

2.4 Tools

The last step of the simplification process of timed automata toward implementation is a little step in which the model of timed safety automata is adapted to several real implementations. The studies in the framework of timed automata and of real-time systems in general have led to the development of several tools for automatic verifications.

The following is an incomplete list of some tools with some references:

KRONOS Performs the model-checking of TCTL formulas with respect to timed safety automata [DOTY96].

UPPAAL Performs the model-checking of properties written in a restricted TCTL logic with respect to slight variants of timed safety automata [BLL+96, LPY97].

HYTECH The HYbrid TECHnology Tool is a tool for the analysis of embedded systems. It computes the condition under which a linear hybrid system satisfied a temporal requirement. Hybrid systems are systems composed of both discrete and continuous components [ACHH93, ACH+95, KPSY93]. Since timed automata are particular hybrid systems they can be verified with this tool [HHW97].

SGM It is a State Graph Manipulator tool for real-time system specification and verification. It uses various sophisticated verification techniques developed in the last years [HW98, WH01].

In this section we want to introduce two automatic tools that will be used in the subsequent chapters to perform automatic verification of the systems used as examples.

2.4.1 KRONOS

KRONOS has been developed at VERIMAG, France. It performs model-checking of timed safety automata with respect to the logic TCTL. Actually, the supported logic is slightly weaker than the one introduced in Section 2.3.5. In particular, the use of the freeze clock operator is restricted. It can only be used for defining bounded operators such as $\forall\Box_{\#c}$ or $\exists\Box_{\#c}$ where $\# \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

The definition of each component of a timed system is given by a description program with a simple syntax that can be used to define the states, the invariants, the boolean variables and the transitions of a timed safety automaton. Although the semantic model is state-based, KRONOS allows labels on the transitions. The synchronization between the components acts as parallel composition of timed safety automata (see Section 2.3.3).

KRONOS can be used to construct the parallel composition of several timed safety automata on which the verification can be performed. With some restrictions, the construction of the parallel composition can be done on-the-fly while performing a checking. This potentially saves space and time.

The verification engine is based on the symbolic model checking approach introduced in [HNSY94]. The details of the implementation can be found in the literature referred in [DOTY96]. An example of analysis using KRONOS can be found in Section 3.6.

2.4.2 UPPAAL

UPPAAL is developed by a team at University of UPPsala in Sweden in collaboration with a team at AALborg University in Denmark.

It is one of the fastest and usable tools in this area. The usability is due to the possibility of specifying the automata graphically and to the existence of a graphical simulator on which some runs can be simulated. The efficiency is due to the fact that it restricts the type of properties that can be checked on those properties that can be reduced to a reachability test. Hence, the verification engine can be better optimized for the task of reachability.

The theoretical papers on which UPPAAL is based analyze the set of properties that can be reduced to reachability and it turns out that many typical real-time requirements can be checked with this restriction [ABBL98, ABG98]. In [LPY95] there is an overview of the algorithmic techniques used in UPPAAL to reduce the state-space explosion on the side of the number of regions and also on the side of parallel composition.

UPPAAL allows to specify a slight variant of timed safety automata. In particular, a system is represented as a network of timed safety automata. This term simply means that the non-synchronization transitions of the components are considered internal actions and that synchronization is modeled by synchronization channels. A synchronization channel can be a normal channel (to specify a synchronization between two timed safety automata) or a broadcast channel (recently supported) that allow synchronization of several components.

Moreover, it is possible to define local and global variables that are not clocks, but integer or boolean variables (or recently arrays). These non-clock variables can be initialized to a value and a test on their value can be inserted in the constraints of a transition. They can be assigned only by the execution of a transition that perform the assignment (as it can reset clocks).

The properties that can be model checked can be expressed by this syntax

$$\begin{aligned} \varphi ::= \;& \exists\Diamond\texttt{Expr} \\ & \mid \forall\Box\texttt{Expr} \\ & \mid \forall\Diamond\texttt{Expr} \\ & \mid \exists\Box\texttt{Expr} \\ & \mid \forall\Box(\texttt{Expr} \rightarrow \exists\Diamond\texttt{Expr}) \end{aligned}$$

where `Expr` can be a boolean expression involving variables or a dot expression of the form $P.s$ that is satisfied only if the component P is in state s.

We refer to the online documentation and to the literature referred at `http:\\www.uppaal.com` for a more precise description of the features of the tool.

Two examples of analysis using UPPAAL can be found in Sections 4.7 and 5.7.

Chapter 3

Non-Instantaneous Actions

Abstract

In this chapter we propose *timed automata with non-instantaneous actions*, a model that allows representing in a suitable way real-time systems. Timed automata with non-instantaneous actions extend the timed automata model by dropping the assumption that actions are instantaneous, that is an action can take some time to be completed. Thus, for an action σ, there are two particular time instants, the *initiation of the action* and the *completion of the action*, which may occur at different times. We investigate the expressiveness of the new model, comparing it with classical timed automata. In particular, we study the set of timed languages which can be accepted by timed automata with non-instantaneous actions. We prove that timed automata with non-instantaneous actions are more expressive than timed automata and less expressive than timed automata with ϵ transitions. Moreover, we define the parallel composition of timed automata with non-instantaneous actions. We show how real systems can be more clearly and suitably modeled by them specifying the railway cross system of Section 2.1. We point out how the specification by means of a parallel composition of timed automata with non-instantaneous actions is, in some cases, more convenient to represent reality. To reach effectiveness for automatic verification we define a transformation to represent a parallel composition of timed automata with non-instantaneous actions by a timed automaton with ϵ transitions. However, with some approximations, the parallel composition can be modeled also by timed safety automata. We reduce to this case and prove the properties of the railway cross system with KRONOS.

This contribution was originally proposed in [BDT00]. A revised version has been published in [BDT01].

When considering real systems in many cases the events are not instantaneous, but have a duration. In this chapter we propose a model, *timed automata with non-instantaneous actions*, which extends the timed automata model by dropping the assumption that Σ-transitions are instantaneous: in our model an action can take some time to be completed. Throughout this chapter we refer often to actions rather than events. This is because events are generally instantaneous while the actions performed by the modeled system usually have a duration. We remark that in the proposed model instantaneous transitions can be modeled as well.

The new model allows specifying in a natural way systems in which the actions have a duration. To model non-instantaneous actions, every edge of the automaton is equipped with two constraints, an *initiation constraint* and a *completion constraint*. An edge can be taken when its initiation constraint is satisfied by the current value of the clocks and it can be completed only when its completion constraint is satisfied. Analogously, every edge is associated with a set of clocks which are reset to zero when the action is initiated (*initiation reset*) and a set of clocks which are reset to zero when the action is completed (*completion reset*).

An action with a duration can be modeled, using timed automata, with two actions, corresponding to the initiation and the completion of the event. However, in this way, the resulting automaton has a different alphabet with respect to the original one and the information that the event is unique is lost.

The notion of timed language accepted by a timed automaton is redefined in the context of timed automata with non-instantaneous actions. Different notions of language acceptance will be defined, which differ in the choice of when an action occurrence must be considered, either on its initiation or on its completion. The different acceptance conditions correspond to different views of the system, on which different properties can be considered: for example, some property may refer to the initiation of an action and the completion of another one.

In Section 3.2 we investigate the expressiveness of the new model, comparing it with classical timed automata. A main result is that timed automata with non-instantaneous actions are strictly more expressive than timed automata and strictly less expressive than timed automata with ϵ edges.

Afterward, we define the parallel composition of timed automata with non-instantaneous actions in Section 3.3. The key idea is that synchronization actions with a duration must be taken synchronously by all the participating components for all the duration of the action. On the other hand, independent actions from different components can be executed in parallel allowing their interval of duration to overlap. In other words, this means that during an execution of an action of one component other actions can be initiated or completed by other components. This kind of behavior is hard to define with a syntactical definition as done for timed automata in Section 1.4.2. We provide a semantic characterization and then we show, in Section 3.5 how to construct a timed automaton with ϵ-transitions that is equivalent to the parallel composition of some given timed automata with non-instantaneous actions. This construction makes effective the use of timed automata with non-instantaneous actions in automatic verification tasks.

Regarding this last aspect, in Section 3.4 the train example presented in Section 2.1 is modeled using non-instantaneous actions. Then, in Section 3.6 the example is adapted to the model of Timed Safety Automata (see Section 2.3.1). The tool

KRONOS is then used to verify automatically the properties of the railway cross system.

3.1 The model

We extend the class $\mathcal{T}$ of timed automata with Büchi acceptance condition in the following way.

Definition 3.1 (Timed Automata with non-Instantaneous Actions) *A timed automaton with non-instantaneous actions N is a tuple $(Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$, where: Q is a finite set of states, Σ is a finite alphabet of symbols, $\mathcal{E}$ is a finite set of edges, $B \subseteq Q$ is the set of initial states, $R \subseteq Q$ is the set of repeated states, $\mathcal{X}$ is a finite set of clocks.*

Each edge $e \in \mathcal{E}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times Q$. If $e = (q, \psi^i, \gamma^i, \sigma, \psi^c, \gamma^c, q')$ is an edge, then q is the source, *q' is the* target, *ψ^i and ψ^c are the* initiation constraint *and the* completion constraint, *respectively, σ is the* label, *γ^i and γ^c are the* initiation reset *and the* completion reset, *respectively.*

The timed transition table associated to N is the tuple $\widehat{N} = (Q, \Sigma, \mathcal{E}, B, \mathcal{X})$.

The class of all timed automata with non-instantaneous actions will be denoted by $\mathcal{N}$. Automata in this class are ranged over by $N, N', \ldots, N_1, N_2, \ldots$.

The timed transition table of a timed automaton with non-instantaneous action differs from a timed transition table of a timed automaton only in the definition of edges. The timed transition system used to define derivations and runs is consequently modified as follows. The states of $\mathcal{S}(\widehat{N})$ are of two kinds:

- a pair (q, ν), where $q \in Q$ is a state of N and $\nu \in \mathcal{V}_{\mathcal{X}}$ is a clock valuation;
- a pair $(\overrightarrow{\sigma}_{(\psi,\gamma,q)}, \nu)$ where $\sigma \in \Sigma$, $\psi \in \Psi_{\mathcal{X}}$, $\gamma \in \Gamma_{\mathcal{X}}$, $q \in Q$ and $\nu \in \mathcal{V}_{\mathcal{X}}$. These states represent the execution of action σ after its initiation.

For an action σ, there are two particular time instants, the *initiation of the action* and the *completion of the action*, which may occur at different times. Following the notation of [JM87] we use $\sigma\uparrow$ to denote the initiation and $\sigma\downarrow$ to denote the completion. The transitions of $\mathcal{S}(\widehat{N})$ are labeled either by a real number representing the elapsed time, or by an initiation or a completion of an action in Σ.

The rules to derive the transitions of $\mathcal{S}(\widehat{N})$ are defined in Figure 3.1.

Rule N1 represents the case in which the automaton is not executing any action and let the time to elapse. The execution of an edge $e = (q, \psi^i, \gamma^i, \sigma, \psi^c, \gamma^c, q')$ is modeled by a transition of $\mathcal{S}(\widehat{N})$ from a state (q, ν) to the state $(\overrightarrow{\sigma}_{(\psi^c,\gamma^c,q')}, \nu\backslash\gamma^i)$ in which σ is initiated (Rule N2). Note that this state records the completion constraint and the completion reset of e, to be considered for the completion of σ. In this state some time can elapse: in this case $\mathcal{S}(\widehat{N})$ reaches a state where σ

N1	$\dfrac{\delta \in \mathbb{R}^{>0}}{(q,\nu) \xrightarrow{\delta} (q, \nu+\delta)}$
N2	$\dfrac{(q,\psi^i,\gamma^i,\sigma,\psi^c,\gamma^c,q') \in \mathcal{E}, \nu \models \psi^i}{(q,\nu) \xrightarrow{\sigma\uparrow} (\overrightarrow{\sigma}_{(\psi^c,\gamma^c,q')}, \nu\backslash\gamma^i)}$
N3	$\dfrac{\delta \in \mathbb{R}^{>0}}{(\overrightarrow{\sigma}_{(\psi,\gamma,q)}, \nu) \xrightarrow{\delta} (\overrightarrow{\sigma}_{(\psi,\gamma,q)}, \nu+\delta)}$
N4	$\dfrac{\nu \models \psi}{(\overrightarrow{\sigma}_{(\psi,\gamma,q)}, \nu) \xrightarrow{\sigma\downarrow} (q, \nu\backslash\gamma)}$

Figure 3.1: Rules for the transitions of $\mathcal{S}(\widehat{N})$

continues to be executed, but the valuation of clocks is modified according to the elapsed time (Rule N3). When the execution of σ terminates, $\mathcal{S}(\widehat{N})$ reaches a new state whose first component is the target state of e and the clock valuation results from the completion reset of e (Rule N4).

As for timed automata with alphabet Σ, the languages accepted by timed automata with non-instantaneous actions $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ are timed languages over Σ. The difference is that here we have different acceptance notions: we can choose, for each action $\sigma \in \Sigma$, if we want to consider the time of its initiation or the time of its completion. If, for some action σ, we choose to consider its initiation, σ is considered as occurring when $\sigma\uparrow$ occurs, and $\sigma\downarrow$ is ignored, while, if we choose to consider its completion, σ is considered as occurring when $\sigma\downarrow$ occurs, and $\sigma\uparrow$ is ignored.

Given $A \subseteq \Sigma$, let us denote by $A\uparrow = \{\sigma\uparrow \mid \sigma \in A\}$ and $A\downarrow = \{\sigma\downarrow \mid \sigma \in A\}$ the set of initiations and completions of the actions in A, respectively.

Definition 3.2 (Selected Sequences)
Let $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton with non-instantaneous actions and let $r = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \dots$ be an infinite derivation *of the timed transition system $\mathcal{S}(\widehat{N})$ such that $s_0 = (q, \nu)$, $q \in B$ and $\nu(x) = 0$ for every $x \in \mathcal{X}$. The* time sequence *and the* label sequence *of r are defined as in Definition 1.3. Moreover,*

- *Given a partition $\Sigma = (I, C)$, $I \cup C = \Sigma$ and $I \cap C = \emptyset$, the (I, C)* selected action sequence *of r is the projection of the event sequence of r on the pairs $\{(l, t) | l \in I\uparrow \cup C\downarrow\}$.*
- *The* state sequence *of r is the projection of the sequence of states $s_0\, s_1\, s_2 \cdots$ of r on the states $\{q_i \in Q \mid s_i = (q_i, \nu_i),\ i = 0 \vee (i \geq 1 \wedge l_{i-1} \in \Sigma\downarrow)\}$*

- *The set of* infinitely many repeated states *of r is denoted by $inf(r)$ and is the set of locations $q \in Q$ that occurs infinitely many times in the state sequence of r.*

Note that the state sequence, used in the following to define acceptance with the Büchi acceptance condition, is composed of states $q \in Q$ of N because a new occurrence in the state sequence is added only after a completion of a Σ-transition.

Definition 3.3 (Runs) *Let $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton with non-instantaneous actions and let r be an infinite* derivation *of the timed transition system $\mathcal{S}(\widehat{N})$. Then r is called a* run *of N iff*

- *r is a fair derivation (Definition 1.4 applied to derivations of $\mathcal{S}(\widehat{N})$)*
- *$inf(r) \cap R \neq \emptyset$*

Now we define the ways in which a timed automaton with non-instantaneous actions can accept timed words on Σ.

Definition 3.4 (Selection, Initiation and Completion Acceptance) *Let $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton with non-instantaneous actions, and let (I, C) be a partition of Σ.*

- *A timed word w over Σ is (I, C)* selection accepted *by N if a run r of N exists such that, for all $\sigma \in I, \sigma' \in C$, $w = v[\sigma/\sigma\uparrow, \sigma'/\sigma'\downarrow]$, where v is the (I, C) selected action sequence of r and $v[\sigma/\sigma\uparrow]$ denotes the sequence v in which every symbol $\sigma\uparrow \in I\uparrow$ is substituted by σ (analogously for $v[\sigma/\sigma\downarrow]$). The set of timed words (I, C) selection accepted by N is called the (I, C)* selection accepted language *of N and is denoted by $\mathcal{L}^s_{(I,C)}(N)$.*
- *A timed word w over Σ is* initiation accepted *by N iff it is $(\Sigma, \emptyset)$ selection accepted by N. We shall use $\mathcal{L}^i(N)$ for $\mathcal{L}^s_{(\Sigma,\emptyset)}(N)$, to denote the* initiation accepted language *of N.*
- *A timed word w over Σ is* completion accepted *by N iff it is $(\emptyset, \Sigma)$ selection accepted by N. We shall use $\mathcal{L}^c(N)$ for $\mathcal{L}^s_{(\emptyset,\Sigma)}(N)$, to denote the* completion accepted language *of N.*

The classes of timed languages selection accepted (for any (I, C)), initiation accepted and completion accepted by automata in $\mathcal{N}$ are denoted by $\mathcal{L}^s(\mathcal{N})$, $\mathcal{L}^i(\mathcal{N})$, $\mathcal{L}^c(\mathcal{N})$, respectively.

As an example, consider the automaton N_1 of Figure 3.2.

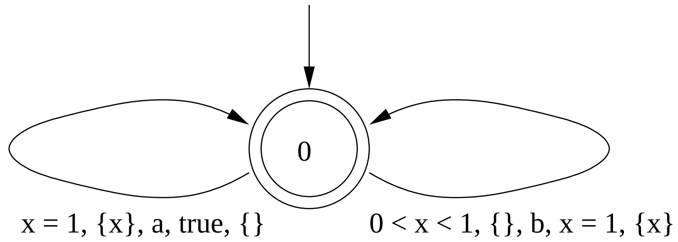

Figure 3.2: Automaton N_1

- Let r be the following derivation of $\mathcal{S}(\widehat{N_1})$: $(0, x = 0) \xrightarrow{0.3} (0, x = 0.3) \xrightarrow{0.7}$ $(0, x = 1) \xrightarrow{a\uparrow} (\overrightarrow{a}_{(true,\emptyset,0)}, x = 0) \xrightarrow{0.2} (\overrightarrow{a}_{(true,\emptyset,0)}, x = 0.2) \xrightarrow{a\downarrow} (0, x = 0.2) \xrightarrow{0.1}$ $(0, x = 0.3) \xrightarrow{b\uparrow} (\overrightarrow{b}_{(x=1,\{x\},0)}, x = 0.3) \xrightarrow{0.7} (\overrightarrow{b}_{(x=1,\{x\},0)}, x = 1) \xrightarrow{b\downarrow} (0, x = 0) \xrightarrow{1}$
  $(0, x = 1) \xrightarrow{a\uparrow} (\overrightarrow{a}_{(true,\emptyset,0)}, x = 0) \xrightarrow{a\downarrow} (0, x = 0) \cdots$
- $(a\uparrow, 1)(b\uparrow, 1.3)(a\uparrow, 3)\cdots$ is the $(\{a, b\}, \emptyset)$ selected action sequence of r. It corresponds to the $(\{a, b\}, \emptyset)$ selection accepted timed word $(a, 1)(b, 1.3)(a, 3)\cdots$
- $(a\downarrow, 1.2)(b\downarrow, 2)(a\downarrow, 3)\ldots$ is the $(\emptyset, \{a, b\})$ selected action sequence of r. It corresponds to the $(\emptyset, \{a, b\})$ selection accepted timed word $(a, 1.2)(b, 2)(a, 3)\cdots$
- $(a\uparrow, 1)(b\downarrow, 2)(a\uparrow, 3)\ldots$ is the $(\{a\}, \{b\})$ selected action sequence of r. It corresponds to the $(\{a\}, \{b\})$ selection accepted timed word $(a, 1)(b, 2)(a, 3)\cdots$

The initiation accepted language of N_1 is the set

$$L_1 = \{(\overline{\sigma}, \overline{t}) \mid \forall i \in \mathbb{N}.\ t_i \in (i, i+1] \wedge ((\sigma_i = a \wedge t_i = i+1) \vee (\sigma_i = b \wedge t_i \in (i, i+1)))\}$$

Consider, now, the automaton N_2 of Figure 3.3. Its completion accepted language is

$$L_2 = \{(\overline{\sigma}, \overline{t}) \mid \forall i \in \mathbb{N}.\ t_i < t_{i+1}, \exists w > 0 \colon \forall i \in \mathbb{N}.\ 0 < t_i < t_0 + 1 - w\}$$

We want to remark that the timed languages L_1 and L_2 are the same languages introduced in Section 1.7 and accepted by the timed automata with ϵ-transitions T_a (Figure 1.15) and T_z (Figure 1.14) respectively. This fact will be used in some proofs below.

3.2 Expressive power of the model

In this section we prove that the expressive power of timed automata with non-instantaneous actions is greater than that of timed automata (without ϵ edges). To

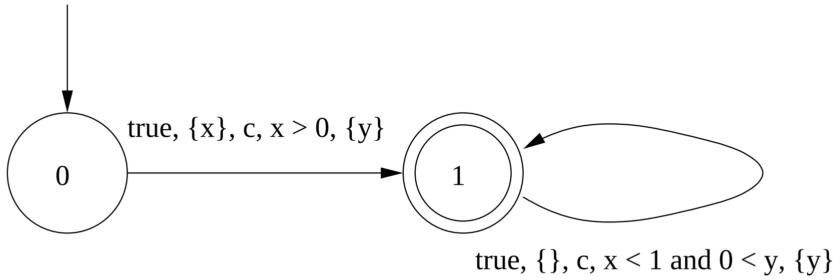

Figure 3.3: Automaton N_2

achieve this result we first introduce a useful device, the simulator automaton, that will be useful to prove the theorems and to define the effective parallel composition in Section 3.5.

Moreover, we state intermediate results on the subclass of initiation and completion accepting timed automata.

3.2.1 Simulation

Definition 3.5 (Simulator Automaton) *Let $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton with non-instantaneous actions. The simulator automaton of N is a timed automaton T^N defined as follows: $T^N = (Q', \Sigma\!\uparrow \cup \Sigma\!\downarrow, \mathcal{E}', B, R, \mathcal{X})$ where $Q' = Q \cup \{q_e |\ e \in \mathcal{E}\}$ and for all $e = (q, \psi^i, \gamma^i, \sigma, \psi^c, \gamma^c, q') \in \mathcal{E}$, $\mathcal{E}'$ contains $(q, \psi^i, \gamma^i, \sigma\!\uparrow, q_e)$ and $(q_e, \psi^c, \gamma^c, \sigma\!\downarrow, q')$.*

The simulator automaton T^N simulates all runs of N retaining both initiation symbol and termination symbol for each action. Its actions are instantaneous and represent initiating instants and terminating instants of non-instantaneous actions of N. In the following, to save notation, we omit the superscript N in T^N when it is clear from the context that T is the simulator automaton of N.

The following definition introduces a useful technical device that we use in the proofs.

Definition 3.6 (Correspondence relation)
Let $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton with non-instantaneous actions and $T^N = (Q', \Sigma\!\uparrow \cup \Sigma\!\downarrow, \mathcal{E}', B, R, \mathcal{X})$ be its simulator automaton and consider their associated timed transition systems $\mathcal{S}(\widehat{N})$ and $\mathcal{S}(\widehat{T^N})$. The correspondence relation *ρ_N between states of $\mathcal{S}(\widehat{N})$ and $\mathcal{S}(\widehat{T^N})$ is defined as follows:*

1. *for all $q \in Q$, $q' \in Q'$ and $\nu \in \mathcal{V}_{\mathcal{X}}$ we let*
   $\rho_N((q, \nu), (q', \nu))$ iff $q = q'$
2. *for all $\sigma \in \Sigma$, $\psi^c \in \Psi_{\mathcal{X}}$, $\gamma^c \in \Gamma_{\mathcal{X}}$, $q' \in Q$, $q_e \in Q'$ and $\nu \in \mathcal{V}_{\mathcal{X}}$ we let*
   $\rho_N((\overrightarrow{\sigma}_{(\psi^c,\gamma^c,q')}, \nu), (q_e, \nu))$ iff q_e *is the state of the simulator automaton T^N*

associated to the transition $e = (q, \psi^i, \gamma^i, \sigma, \psi^c, \gamma^c, q') \in \mathcal{E}$ *of* N *and this non-instantaneous transition is represented in* $\mathcal{S}(\widehat{N})$ *by the state* $(\overrightarrow{\sigma}_{(\psi^c,\gamma^c,q')}, \nu)$.

The following Lemma states the level in which the simulator automaton is equivalent to the main automaton.

Lemma 3.1
Let $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ *be a timed automaton with non-instantaneous actions and* T^N *be its simulator automaton. Let us denote* T^N *by* T.

1. *Let* r^N *be a run of* N, *there exists a run* r^T *of* T *such that the label sequence of* r^T *is equal to the label sequence of* r^N
2. *Let* r^T *be a run of* T, *there exists a run* r^N *of* N *such that the label sequence of* r^N *is equal to the label sequence of* r^T

Proof. We first construct a run of T having the same label sequence of a given run of N and such that every state corresponds to the ones of the run of N.

1. Let $r^N = s_0^N \xrightarrow{l_0^N} s_1^N \xrightarrow{l_1^N} \ldots \xrightarrow{l_{j-1}^N} s_j^N \xrightarrow{l_j^N} s_{j+1}^N \ldots$ be a run of N. Consider the run $r^T = s_0^T \xrightarrow{l_0^T} s_1^T \xrightarrow{l_1^T} \ldots \xrightarrow{l_{j-1}^T} s_j^T \xrightarrow{l_j^T} s_{j+1}^T \ldots$ of T inductively constructed as follows. The initial state is $s_0^T = s_0^N = (q_0, \nu_0)$ with $q_0 \in B$ and $\nu_0(x) = 0$ for every $x \in \mathcal{X}$. Thus r^T start at the same instant and in the same initial state of r^N. Note that $\rho_N(s_0^T, s_0^N)$ holds.

Let j be a natural number. Suppose by induction that $\mathcal{S}(\widehat{N})$ and $\mathcal{S}(\widehat{T})$ are in states s_j^N and s_j^T such that $\rho_N(s_j^N, s_j^T)$. Consider, now, the $(j+1)$-th step of r^N, i.e. $\mathcal{S}(\widehat{N})$ moves to state s_{j+1}^N with label l_j^N. Then $\mathcal{S}(\widehat{T})$ can move to a correspondent state having a transition with the same label:

- if $\mathcal{S}(\widehat{N})$ uses the rule $N1$ to let a time $\delta \in \mathbb{R}^{>0}$ to elapse, then $\mathcal{S}(\widehat{T})$ uses $T1$ to let the same time δ to elapse. Note that, by definition of ρ_N, $\mathcal{S}(\widehat{T})$ reaches a state s_{j+1}^T such that $\rho_N(s_{j+1}^N, s_{j+1}^T)$.
- if $\mathcal{S}(\widehat{N})$ uses the rule $N2$ to start an action then, by induction, we know that $s_j^T = (q, \nu)$ and, by the definition of the simulator automaton, we know that $\mathcal{E}'$ contains a transition $(q, \psi^i, \gamma^i, \sigma\!\uparrow, q_e)$. Thus, $\mathcal{S}(\widehat{T})$ can move with the rule $T2$ to the state $(q_e, \nu\backslash\gamma^i) = s_{j+1}^T$ Note that, again by definition, $\rho_N(s_{j+1}^N, s_{j+1}^T)$.
- if $\mathcal{S}(\widehat{N})$ uses the rule $N3$ we know, by induction, that s_j^T is a correspondent state of s_j^N, so $s_j^T = (q_e, \nu)$ for an certain $e \in \mathcal{E}$. This time $\mathcal{S}(\widehat{T})$ use $T1$ again to reach the state $(q_e, \nu + \delta) = s_{j+1}^T$.

- if $\mathcal{S}(\widehat{N})$ uses the rule $N4$ we know, by induction and by definition of the simulator automaton, that $s_j^T = (q_e, \nu)$ where $\mathcal{E}$ contains $e = (q, \psi^i, \gamma^i, \sigma, \psi^c, \gamma^c, q')$ and $\mathcal{E}'$ contains $(q_e, \psi^c, \gamma^c, \sigma\downarrow, q')$. So $\mathcal{S}(\widehat{T})$ can use the rule $T2$ to reach the state $(q', \nu\backslash\gamma^c) = s_{j+1}^T$. Reached states are correspondent.

Step **2.** is symmetric to step **1.** ■

Definition 3.7 (Relabeled Simulator Automaton)
Let $N = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton with non-instantaneous actions and T^N its simulator automaton. Let (I, C) a partition of Σ.

The renaming function $g_{(I,C)} : (\Sigma\uparrow \cup \Sigma\downarrow) \to (\Sigma \cup \{\epsilon\})$ *is defined as follows:*

$$g_{(I,C)}(\sigma\uparrow) = \begin{cases} \sigma & \text{if } \sigma \in I \\ \epsilon & \text{if } \sigma \in C \end{cases}$$
$$g_{(I,C)}(\sigma\downarrow) = \begin{cases} \epsilon & \text{if } \sigma \in I \\ \sigma & \text{if } \sigma \in C \end{cases}$$

The relabeled simulator automaton *of N, denoted by $T^N_{(I,C)}$, is the simulator automaton T^N where for all $e \in \mathcal{E}'$ the transition label $l \in \Sigma\uparrow \cup \Sigma\downarrow$ of e is renamed by $g_{(I,C)}(l)$.*

Note that the relabeled simulator automaton of N has an alphabet $\Sigma \cup \{\epsilon\}$ and belongs to the class $\mathcal{T}_\epsilon$ (see Section 1.7).

The following Lemma conclude the construction of a simulator for a timed automaton with non-instantaneous actions.

Proposition 3.2 *Let N be a timed automaton with non-instantaneous actions and let (I, C) be a partition of its alphabet Σ. Then $\mathcal{L}^s_{(I,C)}(N) = \mathcal{L}(T^N_{(I,C)})$*

Proof. Let r be a run of N and $(l_0, t_o)(l_1, t_1)\ldots$ be the label sequence of r. Let v be the (I, C) selected action sequence of r. By definition, v is obtained by the projection of the event sequence of r on the pairs $\{(l, t) \mid l \in I\uparrow \cup C\downarrow\}$. The timed word accepted by N through the run r is obtained by a substitution as $\omega = v[\sigma/\sigma\uparrow, \sigma'/\sigma'\downarrow]$ where $\sigma \in I$ and $\sigma' \in C$.

There exists, by Lemma 3.1, a run r' of T^N with the same label sequence of r. There exists also a run r'' of $T^N_{(I,C)}$ such that the label sequence of r'' is $(g_{(I,C)}(l_0), t_0)(g_{(I,C)}(l_1), t_1)\ldots$. The timed word ω' accepted by $T^N_{(I,C)}$ through the run r'' is the projection of the label sequence on the pairs $\{(l, t) \mid l \in \Sigma\}$. Note that in the last projection labels equal to ϵ are not considered.

By the definition of the relabeling function $g_{(I,C)}$ and the substitution above we have $\omega' = \omega$. This shows $\mathcal{L}^s_{(I,C)}(N) \subseteq \mathcal{L}(T^N_{(I,C)})$.

To obtain the language equivalence among N and $T^N_{(I,C)}$ we have to show the converse: $\mathcal{L}(T^N_{(I,C)}) \subseteq \mathcal{L}^s_{(I,C)}(N)$. This can be obtained in the same way using the symmetric result stated in Lemma 3.1. ■

Note that a relabeled simulator automaton $T^N_{(I,C)}$ is a timed automaton with ϵ-transitions, but its particular structure does not allow it to accept finite timed words with infinite fair derivation of the timed transition system $\mathcal{S}(\widehat{T^N_{(I,C)}})$. Thus, we have not to consider the case of finite timed words (see Section 1.7).

3.2.2 Classes inclusions

Now we can relate the expressive power of the class of automata introduced in this chapter with respect to the previous introduced classes.

Theorem 3.3 1. $\mathcal{L}(\mathcal{T}) \subset \mathcal{L}^i(\mathcal{N})$, 2. $\mathcal{L}(\mathcal{T}) \subset \mathcal{L}^c(\mathcal{N})$

Proof. To show that $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}^i(\mathcal{N})$ and $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}^c(\mathcal{N})$, we give a construction that, given a timed automaton T, builds a timed automaton with non-instantaneous actions N such that $\mathcal{L}(T) = \mathcal{L}^i(N) = \mathcal{L}^c(N)$.

Let $T = (Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$ be a timed automaton, the corresponding timed automaton with non-instantaneous actions is $N = (Q, \Sigma, \mathcal{E}', I, R, \mathcal{X}')$, where

- $\mathcal{X}' = \mathcal{X} \cup \{x_0\}$ with $x_0 \notin \mathcal{X}$
- for each $e = (q, \psi, \gamma, \sigma, q') \in \mathcal{E}$, $\mathcal{E}'$ contains $(q, \psi, \gamma \cup \{x_0\}, \sigma, x_0 = 0, \emptyset, q')$

All actions of N are forced to be instantaneous because the fresh clock x_0 is reset at the initiation of the every transition e and it is required that $x_0 = 0$ at the completion of e. This means that N acts as T in all runs, and hence $\mathcal{L}(T) = \mathcal{L}^i(N) = \mathcal{L}^c(N)$ no matter the given selection for N.

To show that the class $\mathcal{L}(\mathcal{T})$ is a strict subset of $\mathcal{L}^i(\mathcal{N})$, we consider the timed automaton with non-instantaneous actions N_1 in Figure 3.2. The language initiation accepted by this automaton, L_1, is equal to the one accepted by the timed automaton with ϵ edges shown in Figure 1.15 of Section 1.7. There, it is recalled that this language cannot be accepted by any timed automaton without ϵ edges (see [BPDG98]).

Analogously, to show that the class $\mathcal{L}(\mathcal{T})$ is a strict subset of $\mathcal{L}^c(\mathcal{N})$, we consider the timed automaton with non-instantaneous actions N_2 in Figure 3.3. The language completion accepted by this automaton is equal to the one accepted by the timed automaton with ϵ edges shown in Figure 1.14 of Section 1.7. Also this language cannot be accepted by any timed automaton without ϵ edges [BPDG98]. ■

Proposition 3.4 1. $\mathcal{L}^i(\mathcal{N}) \not\subseteq \mathcal{L}^c(\mathcal{N})$, 2. $\mathcal{L}^c(\mathcal{N}) \not\subseteq \mathcal{L}^i(\mathcal{N})$

Proof. We show that the language L_1 (L_2), initiation (completion) accepted by the automaton N_1 in Figure 3.2 (N_2 in Figure 3.3), cannot be completion (initiation) accepted by any timed automaton with non-instantaneous actions. To do this, we adapt the proofs given in [BPDG98] to show that L_1 and L_2 cannot be accepted by timed automata without ϵ-transitions.

Part 1. Suppose, for contradiction, that there exists a timed automaton with non-instantaneous actions N_1' that accepts L_1 using the partition $(\emptyset, \{a, b\})$ for the alphabet. Consider the simulator automaton T_1 of N_1'. This automaton is a timed automaton without ϵ-transitions on the alphabet $\{a\uparrow, a\downarrow, b\uparrow, b\downarrow\}$ and we may transform it into a disjunction free and diagonal free one.

We use the notion of precise action and the relative result that have been introduced at the end of Section 1.7. Let $\delta > 0$ and $C_{\max}$ be the constants for this automaton as they have been defined there. By Lemma 3.1 and by our assumption that L_1 is completion accepted by N_1', we can deduce that a timed word of the following form is accepted by T_1:

$$(b\uparrow, t_0^i)(b\downarrow, t_0^c)\ (b\uparrow, t_1^i)(b\downarrow, t_1^c)\cdots(b\uparrow, t_{d-1}^i)(b\downarrow, t_{d-1}^c)\ (a\uparrow, t_d^i)(a\downarrow, t_d^c)\cdots$$

where, $t_{d-1}^c < t_d^i \leq t_d^c = d \in \mathbb{N}$, $0 \leq t_0^i < t_0^c < 1$ and $t_{j-1}^c \leq t_j^i < t_j^c < j+1$ for all $0 < j < d$.

We can choose $d > C_{\max}$, $t_j^c \in (j, j+1) - \delta\mathbb{N}$ for all $0 \leq j < d$ and $t_j^i \in (j, j+1) - \delta\mathbb{N}$ for all $0 \leq j \leq d$.

Let π be a path of T_1 accepting a timed word of this type. Since the action a only occurs at integer times, all occurrences of a should be precise. In particular, the first occurrence of a is precise. Since $d > C_{\max}$, we can deduce from Theorem 1.2 that one of the preceding occurrences of symbols in π should be precise in π. But such a precise action cannot be the any of the occurrence of $b\uparrow$ or $b\downarrow$ or $a\uparrow$ because we have chosen the times $t_j^i, t_j^c \in (j, j+1) - \delta\mathbb{N}$ and any precise action in π, by Theorem 1.2, has to occur in one of the times $\delta\mathbb{N}$. Thus, we have a contradiction.

Part 2. Suppose, again for contradiction, that there exists a timed automaton with non-instantaneous actions N_2' that accepts L_2 using the partition $(\{c\}, \emptyset)$ for the alphabet. As above, consider the simulator automaton of N_2', T_2.

Let C be the finite set of rational constants appearing in the constraints of T_2 and let $t_0 > 0$ be an arbitrary positive real number. We can choose a number δ such that $0 < \delta < 1$, $\delta < \min\{c \in C \mid c > 0\}$ and $(t_0+1-\delta, t_0+1) \cap (C \cup \{t_0 + c \mid c \in C\}) = \emptyset$.

Now, consider a timed word of the following form:

$$(c\uparrow, t_0)(c\downarrow, t_1)(c\uparrow, t_2)(c\downarrow, t_3)\cdots$$

such that $t_0 + 1 - \delta < t_1 < t_2 < t_3 < \cdots$ and $\lim_{i\to\infty} t_i = t_0 + 1 - w$ for some $0 < w < \delta$.

By our assumption that N_2' completion accepts L_2 and by Lemma 3.1, we have that a timed word of the type above has to be accepted by T_2.

This means that there exists a path in T_2 along which this timed word is accepted. However, we can deduce, by the choice of δ, that along the same path also the following timed word can be accepted:

$$(c\uparrow, t_0)(c\downarrow, t_1 + w)(c\uparrow, t_2 + w)(c\downarrow, t_3 + w)\cdots$$

But we have that $\lim_{i\to\infty} t_i + w = t_0 + 1$ and this means that N_2' accepts also a timed word which is not in L_2. This is a contradiction. ■

Proposition 3.5 $\mathcal{L}^i(\mathcal{N}) \cup \mathcal{L}^c(\mathcal{N}) \subset \mathcal{L}^s(\mathcal{N})$.

Proof. The part $\mathcal{L}^i(\mathcal{N}) \cup \mathcal{L}^c(\mathcal{N}) \subseteq \mathcal{L}^s(\mathcal{N})$ follows easily from Definition 3.4.

For the strict inclusion consider the language $L_1 \cup L_2$ where L_1 and L_2 are the languages introduced at the end of Section 3.1. It is easy to see that $L_1 \cup L_2$ is $(\{a, b\}, \{c\})$ selection accepted by the automaton N_3 in Figure 3.4 and, thus, that it belongs to the class $\mathcal{L}^s(\mathcal{N})$. We show in the following that it does not belong neither to $\mathcal{L}^i(\mathcal{N})$ nor to $\mathcal{L}^c(\mathcal{N})$.

Recall part 1 of the proof of Proposition 3.4. We have shown that there is an infinite set of timed words belonging to L_1, and thus to $L_1 \cup L_2$, such that a timed automaton with non-instantaneous actions cannot exist which *initiation* accepts them. Thus, $L_1 \cup L_2$ cannot belong to $\mathcal{L}^i(\mathcal{N})$.

The other possibility is that $L_1 \cup L_2 \in \mathcal{L}^c(\mathcal{N})$. In this case consider the part 2 of the proof of Proposition 3.4. We have shown that, similarly to the previous case, a timed automaton with non-instantaneous actions cannot exist which *completion* accepts a certain infinite set of timed words belonging to L_2. Therefore, we have also that $L_1 \cup L_2 \notin \mathcal{L}^c(\mathcal{N})$. ■

Theorem 3.6 $\mathcal{L}^s(\mathcal{N}) \subset \mathcal{L}(\mathcal{T}_\epsilon)$.

Proof. First we show that $\mathcal{L}^s(\mathcal{N}) \subseteq \mathcal{L}(\mathcal{T}_\epsilon)$. Consider a timed automaton with non-instantaneous actions $N = (Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$, and a partition (I, C) of Σ. Let T be the simulator automaton of N and $T^N_{(I,C)}$ be the relabeled simulator automaton of N. Recall that this automaton belongs to the class $\mathcal{T}_\epsilon$. By Proposition 3.2, we have that $\mathcal{L}^s_{(I,C)}(N) = \mathcal{L}(T^N_{(I,C)})$.

Thus, any timed automaton with non-instantaneous actions can be simulated by an automaton in $\mathcal{T}_\epsilon$.

To show that the class $\mathcal{L}^s(\mathcal{N})$ is a strict subset of $\mathcal{L}(\mathcal{T}_\epsilon)$ (and then also $\mathcal{L}^i(\mathcal{N})$ and $\mathcal{L}^c(\mathcal{N})$ are so, by Proposition 3.5) consider the timed automaton in $\mathcal{T}_\epsilon$ shown in Figure 1.13 of Section 1.7. Recall that the language accepted by this automaton is

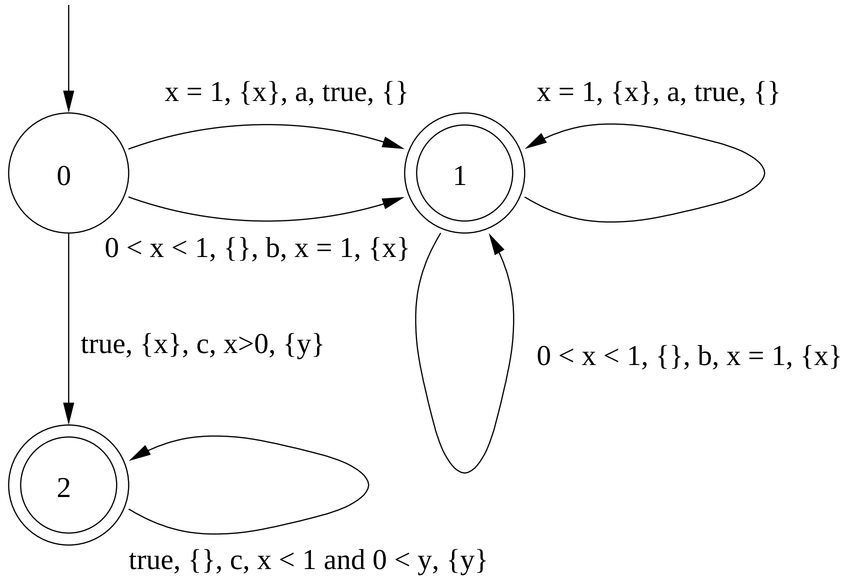

Figure 3.4: Automaton N_3

$$L_{even} = \{(a^\omega, \bar{t}) \mid \forall i \in \mathbb{N}.\ t_i \leq t_{i+1} \wedge (\exists h \in \mathbb{N}\colon\ t_i = 2h)\}$$

We show that this language cannot be selection accepted by any automaton in $\mathcal{N}$. Suppose, for contradiction, that there exists a timed automaton with non-instantaneous actions $N_{even} = (Q, \{a\}, \mathcal{E}, B, R, \mathcal{X})$ accepting this language. Let c be the greatest constant to which the clocks are compared in the clock constraints of $\mathcal{E}$ and let $h \in \mathbb{N}$ such that $2h > c$. Given $k \in \mathbb{R}^{>0}$ let us denote by ν_k a clock valuation such that $\forall x \in \mathcal{X}.\ \nu_k(x) = k$.

Consider a timed word of L_{even} of the form $(a, 2h)\cdots$. N_{even} has to accept it given a partition $(\{a\}, \emptyset)$ or $(\emptyset, \{a\})$. Thus, there exists an edge in $\mathcal{E}$ of the form $(q_0, \psi_0^i, \gamma_0^i, a, \psi_0^c, \gamma_0^c, q_1)$ with $q_0 \in B$. Moreover, there exists a run of N_{even} of the form
$(q_0, \nu_0) \xrightarrow{\delta_0} (q_0, \nu_0 + \delta_0) \xrightarrow{\delta_1} \cdots \xrightarrow{\delta_n} (q_0, \nu_0 + \sum_{j=0}^{n} \delta_j) \xrightarrow{a\uparrow} (\overrightarrow{a}_{(\psi_0^c, \gamma_0^c, q_1)}, (\nu_0 + \sum_{j=0}^{n})\backslash\gamma_0^i)$ $\xrightarrow{\delta'_0} (\overrightarrow{a}_{(\psi_0^c, \gamma_0^c, q_1)}, ((\nu_0 + \sum_{j=0}^{n})\backslash\gamma_0^i) + \delta'_0) \xrightarrow{\delta'_1} \cdots \xrightarrow{\delta'_m} (\overrightarrow{a}_{(\psi_0^c, \gamma_0^c, q_1)}, ((\nu_0 + \sum_{j=0}^{n})\backslash\gamma_0^i) + \sum_{j=0}^{m} \delta'_j) \xrightarrow{a\downarrow} (q_1, (((\nu_0 + \sum_{j=0}^{n})\backslash\gamma_0^i) + \sum_{j=0}^{m} \delta'_j)\backslash\gamma_0^c) \cdots$
where $n, m \geq 0$ and the times δ are positive real numbers.

If N_{even} uses initiation accepting, then we know that $\sum_{j=0}^{n} \delta_j = 2h$ and that $\nu_{2h} = \nu_0 + \sum_{j=0}^{n} \delta_j \models \psi_0^i$. Indeed, given any $p, q \in \mathbb{R}^{>0}$, if $p, q > c$ then $\nu_p \models \psi_0^i$ if and only if $\nu_p \models \psi_0^i$. Thus we have that $\nu_{2h+1} \models \psi_0^i$. Using this fact we can construct another run of N_{even} obtained by the one above letting another time unit elapse before the transition labeled $a\!\uparrow$. Therefore, the timed word $(a, 2h{+}1)\cdots$, not belonging to L_{even}, is initiation accepted by N_{even}. This is a contradiction because we are supposing that its initiation accepted language is L_{even}.

The case in which N_{even} uses completion accepting is similar. ■

As a consequence, our model can be put at an intermediate level between timed automata and timed automata with ϵ edges.

Timed automata with periodic clock constraints, defined in [CG00], also have an expressive power between timed automata and timed automata with ϵ edges. They contain constraints which are based on regularly repeated time intervals. However, their power is not comparable with the power of our model. In fact the aim of automata with periodic clock constraints is that of model periodic behaviors, while we model actions that have a duration.

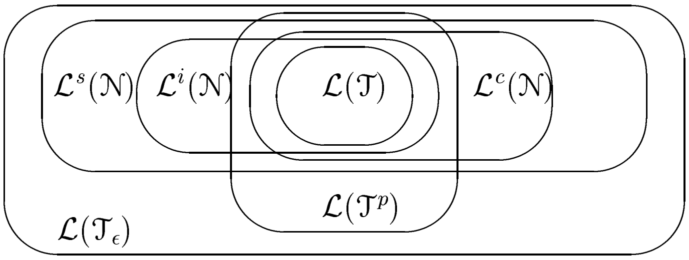

Figure 3.5: Inclusion among language classes

With automata with periodic clock constraints it is possible to model the timed automaton (with ϵ edges) of Figure 1.13 for L_{even}, which we are not able to model. On the other side, automata N_1, N_2 and N_3 cannot be modeled with periodic clock constraints.

The inclusion among language classes is given in Figure 3.5, where $\mathcal{L}(\mathcal{T}^p)$ is the class of languages accepted by timed automata with periodic clock constraints.

3.3 Parallel Composition

To make the definition of the class of timed automata with non-instantaneous actions useful for the specifications of real-time systems we have to define a composition operation for modular specification. Our aim is to extend the definition of parallel composition of timed automata (see Section 1.4.2) to the case of actions with duration. In particular we want to preserve the same notion of synchronization.

It is easy to see that, when transitions can have a duration and there are several components acting in parallel it could happen that, during the execution of an action by a component, another component start the execution of another action. However, this should happen only if the action started by the second component is not a synchronization action with the first component. Moreover, the second component should not be involved in the execution of the transition of the first one.

The synchronization actions are, as in the instantaneous case, those that belong to the alphabet of different components and the rule is that a synchronization action with duration has to be initiated and completed by all the involved components in the same instants. While these components are executing the synchronization action other non-involved components can initiate and/or complete other actions.

It is difficult to specify such a behavior using a syntactical operation as we have done for the parallel composition of timed automata. The main problem is that we should specify states in which some actions are in execution and some others are not. Moreover, when an action is in execution, the state should give the possibility to other independent actions to start or to stop. The syntactic structure of timed automata with non-instantaneous actions given in Definition 3.1 is not suitable to model these behaviors, considering also that the definition of the accepted language depend on a partition of the alphabet. In addiction, recall that we have to assure that in a parallel composition all the timed automata with non-instantaneous actions that are in parallel have a run with their Büchi acceptance condition.

It is an open problem to give the definition of a syntactical parallel composition operation with the characteristics outlined above. We avoid this problem by specifying directly by the timed transition system that would be associated to timed transition table of the result of such a syntactical operator. Then, in Section 3.5, we give a characterization of the timed language accepted by the parallel composition of automata of the class $\mathcal{N}$ by an automaton of the class $\mathcal{T}_\epsilon$.

Let $N_1 = (Q_1, \Sigma_1, \mathcal{E}_1, B_1, R_1, \mathcal{X}_1)$ and $N_2 = (Q_2, \Sigma_2, \mathcal{E}_2, B_2, R_2, \mathcal{X}_2)$ be two timed automata with non-instantaneous actions such that $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. The *parallel composition* of N_1 and N_2 is denoted by $N_1 \parallel N_2$.

We use the notation $\mathcal{S}(\widehat{N_1 \mid N_2})$ to denote a timed transition system which is an extension of the synchronized product between timed transition tables that was introduced in Section 1.3.5. It handles the interleaving of transitions with duration, but it does not impose the respect of the Büchi acceptance conditions of the two components. In other words, it may happen that fair derivations of this transition system could not be runs of the parallel composition $N_1 \parallel N_2$.

The states of $\mathcal{S}(\widehat{N_1 \mid N_2})$ are pairs $\langle \mathcal{C}_1, \mathcal{C}_2 \rangle$ in which $\mathcal{C}_1$ is a state of the timed transition system $\mathcal{S}(\widehat{N_1})$ and $\mathcal{C}_2$ is a state of the timed transition system $\mathcal{S}(\widehat{N_2})$. An initial configuration is $\langle \mathcal{C}_1^0, \mathcal{C}_2^0 \rangle$ in which the components are initial states of $\mathcal{S}(\widehat{N_1})$ and $\mathcal{S}(\widehat{N_2})$ respectively.

To save notation let us define a function $\mathcal{F}_{\text{sync}} : (\Sigma_1 \cup \Sigma_2) \to \{\{1,2\}, \{1\}, \{2\}\}$ such that $\mathcal{F}_{\text{sync}}(\sigma) = \{k \in \{1,2\} \mid \sigma \in \Sigma_k\}$. This function returns a non-singleton if applied to a synchronization action of N_1 and N_2, otherwise it returns a singleton containing the index of the automaton the argument belongs to.

The system generates derivations starting from an initial configuration using the rules showed in Figure 3.6.

Rule $PN1$ represents the case in which both the two automata stay idle while the time passes. Rule $PN2$ describes the situation in which a set of automata initiate, at the same time, an action σ. The set J is $\{1,2\}$ if σ is a synchronization action. If $\mathcal{F}_{\text{sync}}(\sigma)$ is a singleton, only one automaton proceeds according to its own behavior.

PN1	$$\frac{\delta \in \mathbb{R}^{>0}}{\langle(s_1,\nu_1),(s_2,\nu_2)\rangle \xrightarrow{\delta} \langle(s_1,\nu_1+\delta),(s_2,\nu_2+\delta)\rangle}$$
PN2	$$\frac{\mathcal{F}_{\mathrm{sync}}(\sigma) = J, \quad \forall j \in J.\ (\mathcal{C}_j = (q_j,\nu_j),\ (q_j,\psi^i_j,\gamma^i_j,\sigma,\psi^c_j,\gamma^c_j,q'_j) \in \mathcal{E}_j,\ \nu_j \models \psi^i_j)}{\langle \mathcal{C}_1,\mathcal{C}_2\rangle \xrightarrow{\sigma\uparrow} \langle \mathcal{C}'_1,\mathcal{C}'_2\rangle}$$ where for all $j \in \{1,2\},\ \mathcal{C}'_j = \begin{cases} \mathcal{C}_j & \text{if } j \notin J \\ (\overrightarrow{\sigma}_{(\psi^c_j,\gamma^c_j,q'_j)},\nu_j \backslash \gamma^i_j) & \text{if } j \in J \end{cases}$
PN3	$$\frac{\mathcal{F}_{\mathrm{sync}}(\sigma) = J,\ \forall j \in J.\ (\mathcal{C}_j = (\overrightarrow{\sigma}_{(\psi^c_j,\gamma^c_j,q_j)},\nu_j),\ \nu_j \models \psi^c_j)}{\langle \mathcal{C}_1,\mathcal{C}_2\rangle \xrightarrow{\sigma\downarrow} \langle \mathcal{C}'_1,\mathcal{C}'_2\rangle}$$ where for all $j \in \{1,2\}, \mathcal{C}'_j = \begin{cases} \mathcal{C}_j & \text{if } j \notin J \\ (q_j,\nu_j \backslash \gamma^c_j) & \text{if } j \in J \end{cases}$

Figure 3.6: Rules for the timed transition system $\mathcal{S}(\widehat{N_1 \mid N_2})$

Rule $PN3$ describes the case in which a set of automata in parallel, or a single, complete an action σ. Note that synchronization of non-instantaneous actions must be initiated and terminated in the same instants by the two automata.

The definition above can be extended to the case of n, $n \geq 2$ automata simply adding components to the states of the transition system and extending the function $\mathcal{F}_{\text{sync}}$ to handle common symbols of n automata. That is, the function applied to a symbol returns the set of all indexes of automata that must synchronize on the symbol.

Give a derivation r of the transition system $\mathcal{S}(\widehat{N_1 \mid N_2})$, the label sequence and the time sequence of r is defined as in Definition 1.3 and, given a partition (I, C) of $\Sigma_1 \cup \Sigma_2$, the selected action sequence and the state sequence of r is defined as in Definition 3.2.

For the definition of the runs we have to recall the projections on the components given in Definition 1.13 for timed automata. It is simple to adapt this definition to the case of the derivations of the timed transition system $\mathcal{S}(\widehat{N_1 \mid N_2})$: it is sufficient to neglect the counter k on each state used to assure the fairness of the $\|$ operator for timed automata.

We say that a derivation r of $\mathcal{S}(\widehat{N_1 \mid N_2})$ is a run of $N_1 \parallel N_2$ if and only if the derivation $r_{|1}$ is a run of N_1 and $r_{|2}$ is a run of N_2.

With this notion of run, the accepted language of a parallel composition $N_1 \parallel N_2$ is defined as for timed automata with non-instantaneous actions (Definition 3.4).

3.4 An example of specification

Recall the railway cross example specified in Section 2.1. In this section we re-model the system by using timed automata with non-instantaneous actions.

The automaton modeling the behavior of trains is shown in Figure 3.7. Note that the signals `approach` and `exit` are forced to be instantaneous as in the original formulation, while the action `crossing` models in a more suitable way the behavior of trains. It has a duration of at least 1 minute. Note that, because this action is non-instantaneous, in the original formulation it was modeled by two actions, `in` and `out`, simulating the initiation and completion of it.

The gate is modeled by the automaton in Figure 3.8. The synchronization signals `lower` and `raise` are still instantaneous. In this case the specification is more clear because the the closing action is modeled such that it takes between 1 and 2 minutes to be completed (action `down`). Also the opening action `up` has a duration within 1 and 2 minutes.

Finally, Figure 3.9 shows the controller. It is equivalent to the original formulation.

The whole system is obtained by the parallel composition of the three automata.

We would like to remark that our specification allows us to state properties which describe real constraints better than timed automata (recall the formulation

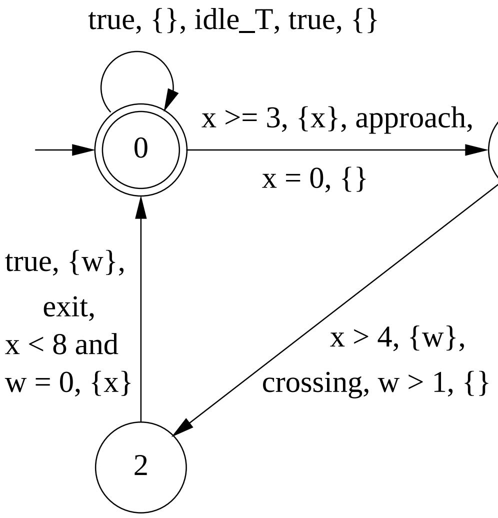

Figure 3.7: Train

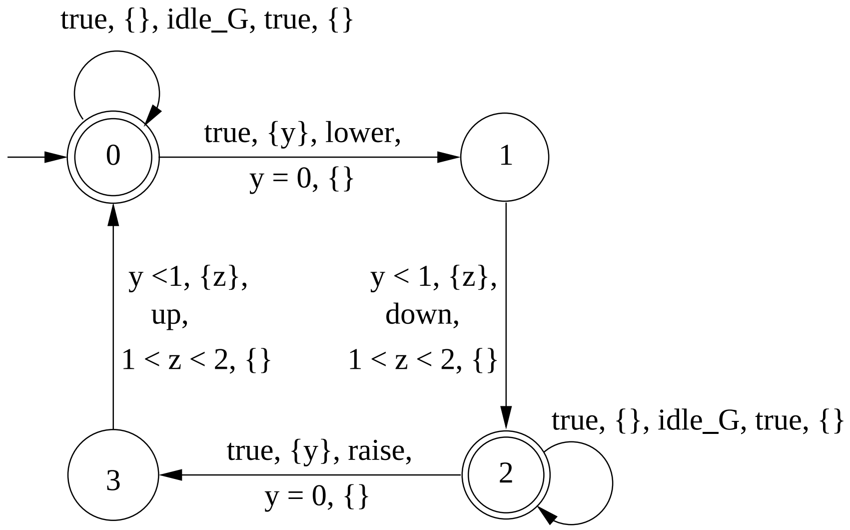

Figure 3.8: Gate

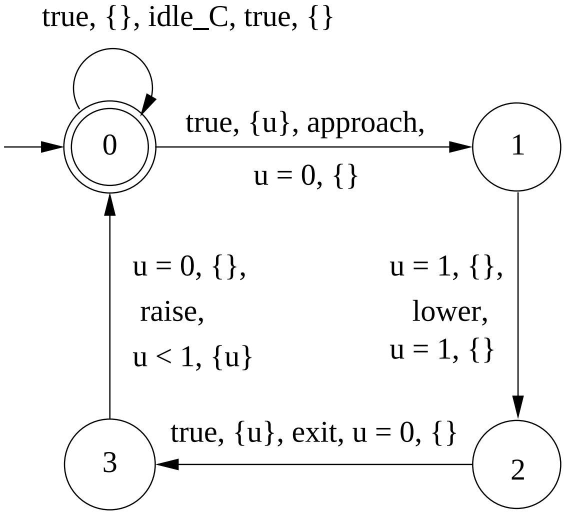

Figure 3.9: Controller

of Section 2.1):

1. "Whenever the train approaches the gate, the gate closes, and when the train *initiates* to cross the gate, the action of *closing* it should be completed".

2. "Every time, after the train crosses the gate, the gate must open within 11 minutes, and the action of opening the gate should be *initiated* only when the train has *completed* the action of crossing it".

Both the properties could be expressed as conditions on the language accepted by the parallel composition (`Train` $\|$ `Gate` $\|$ `Controller`) by specifying that, for property 1., the selection accepted words should refer to the initiation of `crossing` and to the completion of `down`, while for property 2. they should refer to the completion of `crossing` and to the initiation of `up`.

3.5 Simulation for Effectiveness

In this section we define a simple construction that, given a parallel composition $P = (N_1 \parallel N_2)$ of timed automata with non-instantaneous actions and a partition (I, C) of its alphabet $\Sigma_1 \cup \Sigma_2$, builds a timed automaton with ϵ transitions recognizing exactly $\mathcal{L}^s_{(I,C)}(P)$. This is done in order to make the parallel composition effective: properties can be checked on the timed automaton resulting from the construction.

Let $P = (N_1 \parallel N_2)$ is a parallel composition of timed automata with non-instantaneous actions. The construction of the simulator is defined as follows:

1. Construct the simulator automata T^{N_1}, T^{N_2} of N_1, N_2 respectively (see Section 3.2.1).

2. Construct the parallel composition of timed automata $T^P = T^{N_1} \parallel T^{N_2}$ as in Definition 1.12.

3. Construct the relabeled simulator automaton of the automaton T^P, as in Definition 3.7, obtaining the automaton $T^P_{(I,C)}$ which belongs to $\mathcal{T}_\epsilon$.

Clearly, if we have more than 2 automata the construction extends naturally to the general case. First, do step 1 and step 2 for the automata N_1 and N_2. This yields an automaton P'. Do step 1 for the successive automaton N_3 and construct the parallel composition of P' and T^{N_3}. Iterate until all automata have been considered and, finally, do step 3.

Theorem 3.7 $\mathcal{L}^s_{(I,C)}(P) = \mathcal{L}(T^P_{(I,C)})$.

Proof. Lemma 3.1 states that the timed transition system associated to the timed transition table of a simulator automaton generates exactly the same derivations of the one associated to the time transition table of the simulated automaton. Using this fact it is easy to see that the timed transition system $\mathcal{S}(\widehat{N_1 \mid N_2})$ is isomorphic to $\mathcal{S}(\widehat{T^{N_1}} \mid \widehat{T^{N_2}})$. There is a correspondence relation between their states that can be defined on the basis of ρ_{N_1} and ρ_{N_2} (Definition 3.6).

Now, T^{N_1} and T^{N_2} are timed automata having the same Büchi acceptance conditions of N_1 and N_2 respectively. We know, by Proposition 1.1, that the parallel composition operator between timed automata is fair, i.e. the resulting timed automaton accepts only words that result from runs in which all the components have their Büchi acceptance condition satisfied.

Thus, a run r of $T^{N_1} \parallel T^{N_2}$ is such that its image, by the isomorphism, into $\mathcal{S}(\widehat{N_1 \mid N_2})$ is a derivation such that the projections on the original components N_1 and N_2 are runs according to their Büchi acceptance conditions.

This means that the timed words resulting from r belongs to both to $\mathcal{L}^s_{(I,C)}(P)$, by the mechanism of selection accepting, and $\mathcal{L}(T^P_{(I,C)})$, by the renaming function $g_{(I,C)}$. ■

3.6 Analysis with KRONOS

The implementation of timed automata with non-instantaneous actions is straightforward. By implementation we mean a translation into timed safety automata. Since timed safety automata have state-based semantics, the issues regarding ϵ transitions or transitions of the form $\sigma\uparrow$ and $\sigma\downarrow$ are meaningless in this context.

Thus, given a system modeled by a parallel composition of timed automata with non-instantaneous actions we can take the simulator automata of all the components and we can transform them, with the needed approximations (see Section 2.3.2), into timed safety automata. Finally, the parallel composition of these ones is the model on which we can perform automatic verification.

In this section we show this process starting from the parallel composition defined in Section 3.4 and arriving to perform the verification of the properties using KRONOS.

In Figure 3.10 it is shown a timed safety automaton which approximates the behavior of the simulator automaton of the timed automaton with non-instantaneous actions of Figure 3.7. In each state we have inserted the invariant and the boolean variables that are true in it. For each symbol $\sigma \in \Sigma$ we use the symbol $\mathtt{i\_}\sigma$ for $\sigma\!\uparrow$ and $\mathtt{c\_}\sigma$ for $\sigma\!\downarrow$.

Of course, originally instantaneous transitions are not modeled as in the simulator automaton, but they are modeled by a single transition; e.g. the transitions labeled `approach` and `exit`. We have eliminated the idle transition in state 0 and substituted it by the invariant *true*.

In the formulation with timed automata with non-instantaneous actions the fairness condition allows us to specify only the constraint $x < 8$ on the edge labeled `exit` and the constraint $x > 4$ on the edge labeled `crossing`. Without the fairness condition and using the initiation and completion transitions for `crossing` we have to specify, by invariants, the exact times in which the control must stay in each state. In particular we impose that the initiation of the crossing occurs after 4 and before 5 minutes from the approach and that its completion occurs after 1 and before 2 minutes from the initiation.

Note that the automaton of Figure 3.7 has a greater degree of freeness on these times. However, we have to say that, when composed with the other components, the times of occurring of the initiation and the completion of the crossing are forced to be very close to the ones specified above. This is a further evidence that the use of the model of timed safety automata corresponds to a lower, more detailed, view of the system with respect to the specification in which fairness and acceptance conditions are used.

Figure 3.11 shows the Gate and Figure 3.12 the Controller. Similar remarks with respect to the ones that we have done for the Train can be done on these components.

We have specified these timed safety automata with the tool KRONOS. The programs are shown in Figures 3.13, 3.14 and 3.15. All the states of the parallel composition of them are nonZeno. This has been established proving the property

$$(\mathtt{IDLE\_T} \wedge \mathtt{IDLE\_G} \wedge \mathtt{IDLE\_C}) \rightarrow \forall\Box(\exists\Diamond_{=1} true)$$

where $(\mathtt{IDLE\_T} \wedge \mathtt{IDLE\_G} \wedge \mathtt{IDLE\_C})$ represents the initial conditions.

Recall the properties 1. and 2. of Section 3.4. In the system that we have specified with KRONOS we have that the proposition variable `CROSSING` becomes true immediately after a train *initiates* a crossing and it remains true for all the crossing. Immediately after the train *completes* the crossing it becomes false. More-

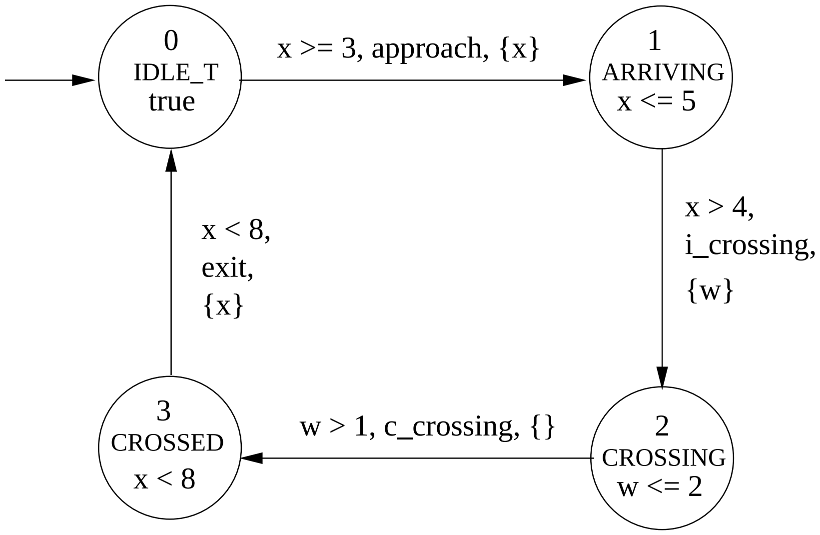

Figure 3.10: Train

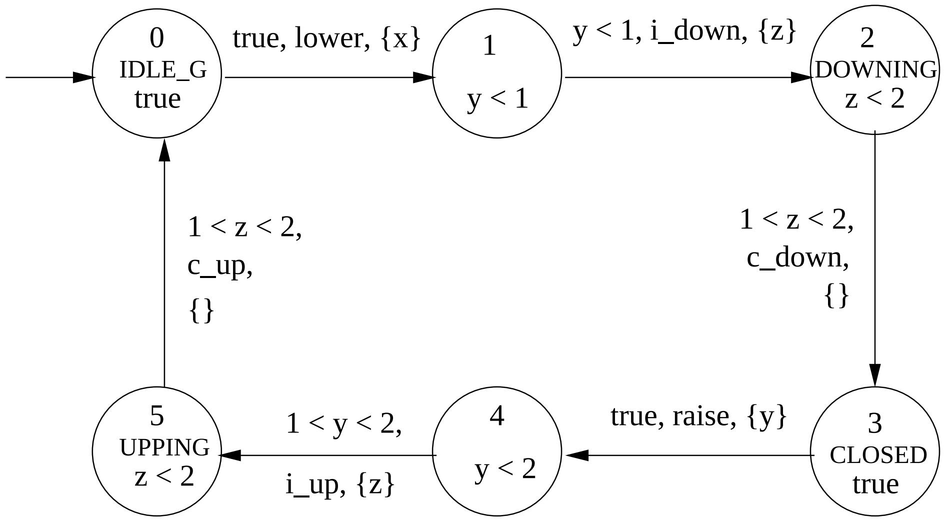

Figure 3.11: Gate

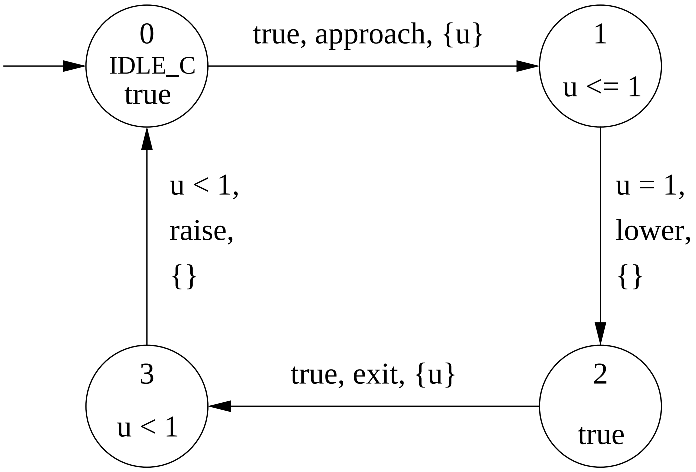

Figure 3.12: Controller

over, we know that `CLOSED` becomes true immediate after the closing of the gate is *completed* and that `UPPING`, as `CROSSING`, is true during the raising of the gate and false otherwise. Thus, the properties can be expressed in TCTL as follows:

1. $(\texttt{IDLE\_T} \wedge \texttt{IDLE\_G} \wedge \texttt{IDLE\_C}) \rightarrow \forall\Box(\texttt{CROSSING} \rightarrow \texttt{CLOSED})$

2. $(\texttt{IDLE\_T} \wedge \texttt{IDLE\_G} \wedge \texttt{IDLE\_C}) \rightarrow \forall\Box(\texttt{CLOSED} \rightarrow \forall\Diamond_{\leq 11}\texttt{IDLE\_G})$
   together with
   $(\texttt{IDLE\_T} \wedge \texttt{IDLE\_G} \wedge \texttt{IDLE\_C}) \rightarrow \forall\Box(\texttt{CROSSING} \rightarrow \neg\texttt{UPPING})$

They are all true. Note that the liveness property requiring the re-opening of the gate within 11 minutes is false if we require an upper bound of 10 minutes.

```
/* Train */

#locs 4
#trans 4
#clocks X W
#sync APPROACH EXIT
loc:0
prop: IDLE_T
invar: TRUE
trans:
X>=3 => APPROACH; X:=0; goto 1
loc: 1
prop: ARRIVING
invar: X <= 5
trans:
X > 4 => I_CROSSING; W:=0; goto 2
loc: 2
prop: CROSSING
invar: W <= 2
trans:
W > 1 => C_CROSSING; ; goto 3
loc: 3
prop: CROSSED
invar: X < 8
trans:
X < 8 => EXIT; x:=0; goto 0
```

Figure 3.13: KRONOS program representing the `Train`

```
/* Gate */

#locs 6
#trans 6
#clocks Y Z
#sync LOWER RAISE
loc: 0
prop: IDLE_G
invar: TRUE
trans:
TRUE => LOWER; Y:=0; goto 1
loc: 1
prop:
invar: Y<1
trans:
y<1 => I_DOWN; Z:=0; goto 2
loc: 2
prop: DOWNING
invar: Z<2
trans:
Z>1 and Z<2 => C_DOWN; ; goto 3
loc: 3
prop: CLOSED
invar: TRUE
trans:
TRUE => RAISE; Y:=0; goto 4
loc: 4
prop:
invar: Y<2
trans:
Y<2 and Y>1 => I_UP; Z:=0; goto 5
loc: 5
prop: UPPING
invar: Z<2
trans:
Z>1 and Z<2 => C_UP; ; goto 0
```

Figure 3.14: KRONOS program representing the `Gate`

```
/* Controller */

#locs 4
#trans 4
#clocks U
#sync APPROACH EXIT LOWER RAISE
loc: 0
prop: IDLE_C
invar: TRUE
trans:
TRUE => APPROACH; U:=0; goto 1
loc: 1
prop:
invar: U<=1
trans:
U=1 => LOWER; ; goto 2
loc: 2
prop:
invar: TRUE
trans:
TRUE => EXIT; U:=0; goto 3
loc: 3
prop:
invar: U<1
trans:
U<1 => RAISE; ; goto 0
```

Figure 3.15: KRONOS program representing the `Controller`

Chapter 4

Urgent Transitions

Abstract

In this chapter we present an extension of the formalism of timed automata by allowing urgent transitions. An urgent transition is a transition which must be taken within a fixed time interval from its enabling time and it has higher priority than other non-urgent transitions enabled in the same state. We give a set of rules formally describing the behavior of urgent transitions and we show that, from a language theoretic point of view, the addition of urgency does not improve the expressive power of timed automata. This is done by defining a transformation that uses first the region construction for timed automata to obtain a timed automaton in region form. Then, the procedure adds transitions and clock constraints to the obtained automaton. The result is a timed automaton without urgent transitions that acts as the original one. From a specification point of view, the use of urgent transitions allows short and clear specifications of behaviors involving urgency and priority. The notion of urgency that we introduce is inherently non-compositional and the transformation procedure that we give to reduce a timed automaton with urgent transitions to a timed automaton introduces an enlargement in the automaton size due to the use of the region construction. We discuss these aspects and show how to make effective our approach using implementable timed automata and standard minimization techniques. To show the features of the proposed extension we specify a multicast protocol for mobile computing using timed automata with urgent transitions and we compare this specification with one that uses the features of the tool UPPAAL.

The material of this contribution was originally presented, in an early version, in [BT01]. The full version has been published in [BT04].

The notion of urgency in timed systems has been already introduced in [BS97, BST98, BS00], where the urgency of transitions outgoing from a state is induced by a time progress condition associated to the state and derived from deadlines associated to the transitions. The semantics imposes the impossibility to stay in the state if the condition is not satisfied while the time elapses. In this Chapter we consider a slightly different notion of urgency. Urgent transitions are transitions which must be performed within a given time interval starting from their enabling and, in this situation, have *priority* upon non-urgent transitions.

From the expressiveness point of view, both our approach and the one of [BS97, BST98, BS00] are suitable for specifying timed systems. The latter allows, in some cases, more general urgency conditions, while, in other cases, such as "as soon as possible" transitions, our approach allows more general time constraints. Moreover, the semantic setting is different. In *op. cit.* the model of timed safety automata (see Section 2.3.1) is used together with time progress conditions in the states. Here we adopt the timed automata model, although we show, in Section 4.5, that our notion of urgency can be defined in the same way for timed safety automata. Section 4.6 contains a more detailed discussion on the differences of the two approaches.

We show that, from the language theoretic point of view, timed automata and timed automata with urgent transitions are equivalent. This is proved by defining a transformation from a timed automaton with urgent transitions to a timed automaton which accepts the same language.

From a specification point of view, urgent transitions provide an easy and effective way to express some important types of behaviors of real time systems, which, otherwise, should be simulated using complex constructions. These usually yield specification that id difficult to understand and they increase the probability of mistakes.

The notions of priority and urgency captured by our definitions are inherently non-compositional and, hence, the transformation is not a congruence with respect to parallel composition. Thus, in the specification task, one has to design components considering that the urgency of their actions can be affected and/or enhanced by other components and by the synchronization mechanism of the parallel composition. This aspect is explained in Sections 4.3 and 4.4. The latter contains an example of specification of a multicast protocol for mobile computing, which is used to illustrate the possible use and the features of timed automata with urgent transitions. Section 4.5 contains some remarks and suggestions for an effective use of timed automata with urgent transitions in the verifications of systems. Finally, Section 4.7 contains an attempt to specify the multicast protocol with the urgency features of UPPAAL, which are weaker than ours.

4.1 Timed automata with urgent transitions

In this section we extend the model of timed automata with a new feature which is useful in the specification of real-time systems. We start from timed Büchi automata (Chapter 1) with the following clock constraints:

$$\begin{aligned}\psi \;::=\;& true \\ & \mid x\#c \\ & \mid \psi \wedge \psi\end{aligned}$$

where x, y are clocks, $\# \in \{<, \leq, >, \geq\}$ and $c \in \mathbb{N}$. Recall Section 1.6 in which we show that this syntax is minimal. Diagonal constraints are not included simply for the sake of simplicity. They can be treated with a slight modification of the transformation given in Section 4.2.

The idea is to provide, in each state of a timed automaton, the possibility of labeling some outgoing edges as *urgent*. Intuitively an urgent labeled edge must be taken with higher priority with respect to the non-urgent ones and it must be taken within a certain time interval starting from its enabling.

Let q be a state of a timed automaton and let e be an outgoing transition from q. We use $\langle b_e^s, b_e^e \rangle$ to denote the time interval in which the transition e is enabled, according to its constraint, in the state q during a derivation of the semantic transition system of the automaton. Note that such an interval is represented by its bounds, b_e^s and b_e^e, which can be closed or open; that is $b_e^s \in \{(t, [t\}$ and $b_e^e \in \{t), t]\}$, where t is, in general, a non-negative rational value.

Let $\ell \in \mathbb{Q}^+$ be a fixed parameter associated to the timed automaton. Let t_q be the time instant in which the state q is entered. Suppose e is an urgent transition. The time interval in which e must be taken starts at time b_e^s, but if the constraint of e is already satisfied when q is entered then the interval starts at time t_q. The interval stops ℓ time units after its start. However, if the constraints of e becomes false before, the interval stops at instant b_e^e. Figure 4.1 shows three cases. The gray-filled areas are the intervals in which e is enabled by its constraints. The oblique-line-filled areas are the intervals in which e is enabled but not executable. The case (a) is the basic case in which the interval has length ℓ, starts at the enabling time of e and stops after ℓ time units while e is still enabled. Case (b) shows the situation in which e becomes disabled before ℓ time units elapsed and the interval is shorter than ℓ. In case (c) the transition is already enabled when the state is entered: the interval starts at t_q and its length is ℓ.

This notion of urgency allows us to define precisely the behavior of urgent actions. The intuitive idea "urgent transitions must be taken as soon as possible" introduces some problems when applied in a model with a dense time domain. To see this, consider a state of a timed automaton in which the current value of clock x is in $[0, 1]$ and there is an outgoing urgent transition with a clock constraint $x > 1$. Letting the time to elapse, at which time should the urgent transition be executed? It is not possible to answer precisely this question since the time domain is dense. To avoid this problem we introduce the constant ℓ and the interval within the action must be executed. Adopting this solution, the problem does not arise and the semantics of urgency is clear in all cases.

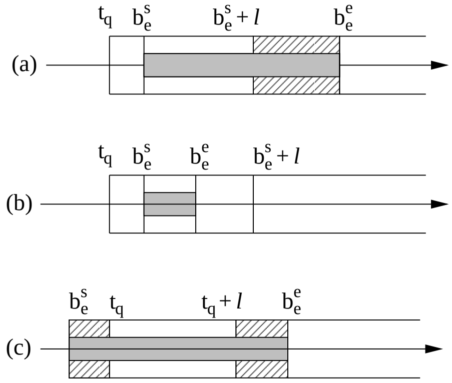

Figure 4.1: The semantics of urgency

To complete the specification of the proposed notion of urgency we have to specify the nature of the bounds of the interval. The choice is arbitrary in principle, but the following seems to be the most natural. If the left bound of the interval is open, then we set the right bound closed, i.e. for the urgent action with constraint $x > 1$ of the example above, the interval in which such an action must be taken is that in which $x \in (1, 1 + \ell]$. Conversely, if the left bound is closed then we set the right bound open. For instance, if the transition has a constraint of the form of $x \geq 1$, instead of $x > 1$, then the interval in which it must be taken is that in which $x \in [1, 1 + \ell)$. When the state is entered and the urgent transition is already satisfied, the left bound of the interval is considered closed. In this way the interval is always well defined.

The denseness also imposes that the constant ℓ be greater than 0. The choice $\ell = 0$ could be interpreted as "immediately", but this leads, in some cases, to the problem discussed above. However, since $\ell \in \mathbb{Q}^+$, it can be chosen as small as needed. In other words, the "as soon as possible" limit behavior can be approximated with arbitrary precision.

Note that the parameter ℓ can be *local* to each urgent transition, but for the sake of simplicity we discuss the case in which ℓ is a global parameter. The case of local specification can be caught by a slight modification of the definition, the semantics and the transformation.

When a state q has a set U_q of outgoing urgent transitions, they are treated as follows. Let b^s be the lower bound of the first interval, in the state, in which some urgent transition becomes enabled. Let $S_q = \{u_1, u_2, \ldots, u_k\}$ be the set of urgent transitions which become enabled after b^s and let $I_{u_i} = \langle b^s_{u_i}, b^e_{u_i} \rangle$, $i = 1, \ldots, k$ be their enabling time intervals. The time interval in which at least one urgent transition must be taken is $\langle b^s, \texttt{min\_bound}(b^s + \ell, b^e_{u_1}, \ldots, b^e_{u_k}) \rangle$, where $\texttt{min\_bound}$ gives the more restrictive interval upper bound. In this interval all enabled urgent transitions have the same priority and one of them is executed non-deterministically. Figure 4.2 shows two possible scenarios. The white area gives the interval in which

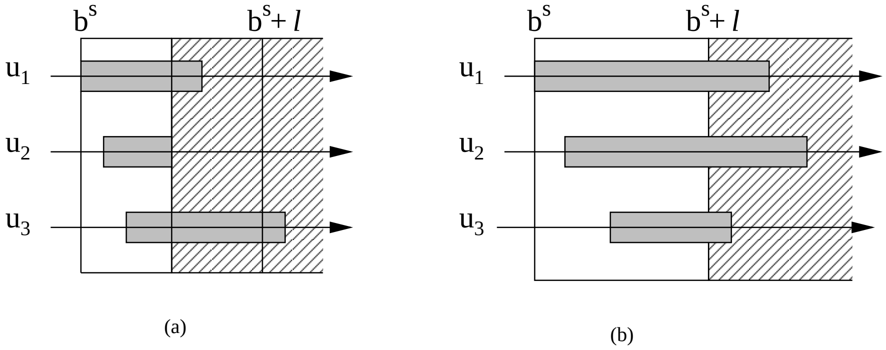

Figure 4.2: The case of more than one urgent transitions overlapping

at least an urgent transition must be taken. In (a) u_1 is the first enabled transition. u_2 has the minimal upper bound, thus the interval in which one urgent transition must be taken starts with $b^s_{u_1}$ and ends with $b^e_{u_2}$. In this interval one of u_1, u_2 and u_3 (in the sub-interval in which it is enabled) can be taken. In (b), the first urgent transition enabled is u_1 and it remains enabled also after ℓ time units. Thus, in this case, the interval to consider has length ℓ because the other ones are still enabled. Within this interval u_2 and u_3 can be taken, when enabled, as well.

If a state has no urgent outgoing edges then the behavior is the usual one of timed automata. This also happens when a state is entered and no urgent transitions are enabled.

Moreover, when some urgent transition is enabled in a state, the unique way to continue the run is to execute it or another urgent transition following the rules expressed above.

Definition 4.1 (Timed Automaton with Urgent Transitions) *Let $\ell \in \mathbb{Q}^+$ be a constant. A timed automaton with urgent transitions $T^\ell_u = (Q, \Sigma, \mathcal{E}, \mathcal{U}, B, R, \mathcal{X})$ is a tuple where Q is a finite set of states, Σ is a finite alphabet of actions, $\mathcal{E}$ and $\mathcal{U}$ are finite sets of edges, the non-urgent and the urgent ones, $B \subseteq Q$ is the set of initial states, $R \subseteq Q$ is the set of repeated states, $\mathcal{X}$ is a finite set of clocks. Each edge $e \in \mathcal{E} \cup \mathcal{U}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times Q$.*

The class of all timed automata with urgent transitions will be denoted by $\mathcal{T}^\ell_u$.

In the following the superscript ℓ could be omitted and, when this happens, it should be considered implicitly defined.

The semantics of a timed automaton with urgent transitions T^ℓ_u is defined, as for timed automata, in terms of its accepted language. This is defined in the same way of the accepted language for timed automata. The difference is that we use a different timed transition system to define the derivations of the timed transition table $\widehat{T^\ell_u}$: the transition system $\mathcal{S}(\widehat{T^\ell_u}) = (S_u, \rightarrow)$. The states S_u are triples (q, ν, δ_q) such that $q \in Q$ is the current state of the automaton T^ℓ_u, ν is the current clock

(Time)	$$\frac{\delta \in \mathbb{R}^{>0}}{(q,\nu,\delta_q) \xrightarrow{\delta} (q,\nu+\delta,\delta_q+\delta)}$$
(Non-Urgent)	$$\frac{(q,\psi,\gamma,\sigma,q') \in \mathcal{E},\ \nu \models \psi, \quad (\forall (q,\psi_u,\gamma_u,\sigma_u,q'_u) \in \mathcal{U}.(\neg\exists\ \delta.\ (0 \le \delta \le \delta_q \wedge \nu - \delta \models \psi_u)))}{(q,\nu,\delta_q) \xrightarrow{\sigma} (q',\nu\backslash\gamma,0)}$$
(Urgent)	$$\frac{\begin{array}{c}(q,\psi_u,\gamma_u,\sigma_u,q') \in \mathcal{U},\ \nu \models \psi_u, \\ (\neg\exists u' = (q,\psi_{u'},\gamma_{u'},\sigma_{u'},q'_{u'}) \in \mathcal{U}. \\ (\exists\delta.0 \le \delta \le \delta_q \wedge \delta \ge \ell \wedge \nu - \delta \models \psi_{u'})), \\ (\neg\exists\ u' = (q,\psi_{u'},\gamma_{u'},\sigma_{u'},q'_{u'}) \in \mathcal{U}. \\ (\exists\delta.0 \le \delta \le \delta_q \wedge \nu \not\models \psi_{u'} \wedge \nu - \delta \models \psi_{u'})\end{array}}{(q,\nu,\delta_q) \xrightarrow{\sigma_u} (q',\nu\backslash\gamma_u,0)}$$

Figure 4.3: Rules for the transitions of $\mathcal{S}(\widehat{T^\ell_u})$

valuation and $\delta_q \in \mathbb{R}^{\ge 0}$ is a number recording the time elapsed since the state q has been entered. The rules to derive the transitions of $\mathcal{S}(\widehat{T^\ell_u})$ are defined in Figure 4.3.

Rule **(Time)** lets the time elapse in a state and updates both the clock valuation and the time elapsed in the state. Recall that, due to the acceptance condition semantics, some states, in the action sequence, must be entered infinitely many times. Thus the automaton is not allowed to elapse time in a state infinitely.

Rule **(Non-Urgent)** can be used when T^ℓ_u is in a state without outgoing urgent edges (the $\forall$ condition is trivially true). In this case the behavior is the same as timed automata. When T^ℓ_u is in a state with a set of urgent outgoing transition, the "$\neg\exists$" condition in the rule requires that every urgent transition has never been enabled since the current state was entered. If this is false the rule is not applicable. Note that when a new state is entered the time elapsed is set to 0.

Rule **(Urgent)** executes an urgent action σ_u. The first "$\neg\exists$" condition ensures that the urgent transition that is going to be executed is taken before a time ℓ has elapsed after the enabling time of any urgent transition. The second "$\neg\exists$" forbids the execution of the urgent transition if there is a urgent transition which was enabled and it is no longer so.

Example 4.2 *Figure 4.4 shows an example of a timed automaton with a urgent* *transition (graphically, we show this by attaching a "u" to the edge). In this example we consider* $\ell = 1$. *The automaton can execute the action b when the value of the clock x is in the interval* $(0, 1]$. *When the value of x becomes greater than* 1*, b cannot be performed any longer and the urgent action a must be executed. Moreover, because of the urgency, a must be performed while the value of x is in the interval* $(1, 2]$.

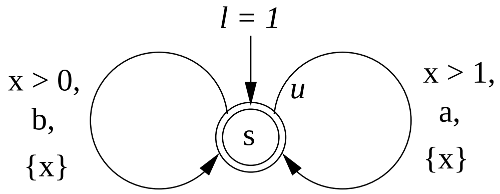

Figure 4.4: An automaton with urgent transitions, T_u^1

4.2 Expressive power of timed automata with urgent transitions

In this section we show that, from a language theoretic point of view, the expressive power of timed automata with urgent transitions is equivalent to the one of timed automata. This is shown by providing a three-steps transformation which preserves the accepted language. Since timed automata are special cases of timed automata with urgent transitions ($\mathcal{U} = \emptyset$), the transformation is only given starting from the latter ones.

4.2.1 The region form of a timed automaton

Let T_u^ℓ be a timed automaton with urgent transitions. We give a transformation that builds a timed automata accepting the same timed language.

Note that if $\ell = \frac{a}{b}$ with a and b natural numbers ($b \neq 0$), it is always possible to transform a $T_u^{\frac{a}{b}}$ automaton to an isomorphic one T_u^a by multiplying all the constants in the clock constraints by b. Practically this means that a different scale is used to measure time and, indeed, this does not affect specification/verification tasks. So we can assume without loss of generality that ℓ is a positive natural number.

In the following we need a transformation of clock constraints that, starting from a constraint ψ, gives a *logically equivalent* constraint $\mathtt{min}(\psi)$ such that it does not contain redundancies. Essentially the transformation drops from ψ the atomic constraints which are implied by others, yielding a minimal conjunction of constraints.

Definition 4.3 *Given a constraint* ψ *over a set* $\mathcal{X}$ *of clocks.* $\mathtt{min}(\psi)$ *is the equivalent constraint where there is only one constraint of the form* $x = c$, $x \# c$, $c \# x \#' d$ *or* $c \# x$, *where* $\#, \#' \in \{<, \leq\}$ *for every clock* $x \in \mathcal{X}$.

Given $x \in \mathcal{X}$, we denote by $\mathtt{select}(\mathtt{min}(\psi), x)$ *such unique constraint for x.*

Recall the region construction for timed transition tables introduced in Section 2.2.1. In particular recall that we denote by $\mathtt{Reg}(\widehat{T})$ the set of all clock regions for a timed transition table $\widehat{T}$ and that, given a clock region $\alpha \in \mathtt{Reg}(\widehat{T})$ and $x \in \mathcal{X}$, we denote by $R_T(\alpha, x)$ the unique clock constraint in C_x in the specification of α. Moreover, given a clock region $\alpha \in \mathtt{Reg}(\widehat{T})$ and a reset $\gamma \subseteq \mathcal{X}$, we denote by $[\gamma \to 0]\alpha$ the clock region such that, for all $x \in \gamma$, the constraint in α for x is substituted by $x = 0$.

In Section 2.2.1 it is shown how to construct, given a clock region $\alpha \in \mathtt{Reg}(\widehat{T})$, the ordered set of clock regions that are *time successors* of α, denoted by $\mathtt{succ}(\alpha)$. The order $\leq_\alpha$ of the clock regions in this set is total and such that $\alpha \leq_\alpha \alpha'$ iff α' is a time successor of α[1].

The following definition describes a first transformation, in region form, of a timed automaton with urgent transitions. To this purpose a state of the transformed automaton records both the state of the original one and the equivalence class (clock region) of the values of clocks when the state is entered. The resulting automaton is a timed automaton that is equivalent to the original one and has a structural property that will be used in the next step of the transformation for proving the correctness of the transformation itself.

Definition 4.4 *Let $T_u = (Q, \Sigma, \mathcal{E}, \mathcal{U}, B, R, \mathcal{X})$ be a timed automaton with urgent transitions. The corresponding timed automaton* in region form*,*
$T_u^r = (Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}^r, B^r, R^r, \mathcal{X})$ is defined as follows:

- *the states in Q^r (resp. R^r) are of the form $\langle q, \alpha\rangle$ where $q \in Q$ (resp. R) and α is a clock region,*

- *the states in B^r are of the form $\langle q, [\nu_0]\rangle$ where $q \in B$ and $\nu_0(x) = 0$ for all $x \in \mathcal{X}$*

- *$(\langle q, \alpha\rangle, \mathtt{min}(\psi) \wedge \bigwedge_{x \in \mathcal{X}} R_{T_u}(\alpha'', x), \gamma, \sigma, \langle q', [\gamma \to 0]\alpha''\rangle) \in \mathcal{E}^r$ (resp. $\mathcal{U}^r$) iff $(q, \psi, \gamma, \sigma, q') \in \mathcal{E}$ (resp. $\mathcal{U}$), $\alpha \in \mathtt{Reg}(\widehat{T_u})$, and $\alpha'' \in \mathtt{succ}(\alpha)$.*

Note that the new states are built exactly as the ones of the region Büchi automaton defined in Definition 2.3. This construction differs from that because constraints and resets are maintained on the edges. These constraints are modified in order to force the corresponding edge to enter only one of the time successor clock regions (in the sense that for other regions the constraint is always false).

Also note that this step can be applied to any timed automaton yielding the following result.

[1]This has to be imposed for those cases in which α does not belong to $\mathtt{succ}(\alpha)$ i.e. points or lines regions.

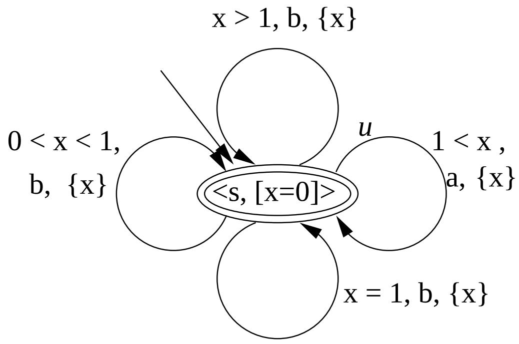

Figure 4.5: Automaton T_u^{1r}

Proposition 4.1 *Given a timed automaton with urgent transitions T_u, let T_u^r be T_u in region form. Then, $\mathcal{L}(T_u) = \mathcal{L}(T_u^r)$.*

Proof. By region construction correctness. ■

We want to remark that the region form of a timed automaton can be, in general, a useful device for reasoning about the automaton itself and its structure. In particular, we exploit, in the next step of the transformation, the following property:

Proposition 4.2 *Let T_u be a timed automaton with urgent transitions and let T_u^r be T_u in region form. In every derivation of the transition system $\mathcal{S}(\widehat{T_u^r})$, if a state $(\langle q, \alpha\rangle, \nu)$ is entered by performing a transition labeled by $\sigma \in \Sigma$, then $[\nu] = \alpha$.*

Proof. By Definition 4.4. ■

As a consequence of this property, note that a clock constraint of the form $x - y \# c$ ($\# \in \{<, \leq, >, \geq\}$) maintains the same truth value while the control is in a state of T_u^r and this truth value is the value of the satisfaction $\alpha \models x - y \# c$ where α is the clock region associated to the state. This means that every diagonal constraint in T_u^ℓ can be substituted in T_u^r by *true* or *false*. Thus, we could allow diagonal constraints in T_u^ℓ as well and eliminate them at this stage of the transformation.

Example 4.5 *In Figure 4.5 it is shown the automaton of Figure 4.4 in region form, denoted by T_u^{1r}. Note that the constraints explicitly show the time successor clock region to which they refer. Note that all the edges with a* false *constraint have been removed and, in the states, there is only the $[x = 0]$ region because both the original edges reset x.*

4.2.2 Making the urgent transitions ℓ-consistent

The second step of the transformation will adapt the constraints of the urgent transitions of T_u^r making them consistent with the semantics we gave in Section 4.1. More precisely clock constraints are adapted according to the behavior expressed by the rule **(Urgent)**. In this step we consider only the urgent actions and neglect the other ones which remain unchanged. The third step will adapt these according to the semantics.

Let $\langle q, \alpha \rangle$ be a state of T_u^r such that in state q of T_u there was a set of outgoing urgent transitions $U_q = \{u_1, \ldots, u_k\}$, where $u_i = (q, \psi_{u_i}, \gamma_{u_i}, \sigma_{u_i}, q'_{u_i})$, $i = 1, 2, \ldots, k$. Each of these transitions becomes, in T_u^r, a set of transitions

$$E_{u_i}^{\alpha} = \{(\langle q, \alpha \rangle, \mathtt{min}(\psi_{u_i}) \wedge \bigwedge_{x \in \mathcal{X}} R_{T_u}(\alpha'', x), \gamma_{u_i}, \sigma_{u_i}, \langle q'_{u_i}, [\gamma_{u_i} \to 0]\alpha'' \rangle) | \\ \alpha'' \in \mathtt{succ}(\alpha)\}$$

We need to determine the minimal upper bound of the enabling interval of all the urgent actions which can be enabled in a state. By the semantics of Section 4.1, we know that urgent transitions must be taken before such a bound.

Using the total order $\leq_\alpha$ defined in the set $\mathtt{succ}(\alpha)$ and the property of states expressed by Proposition 4.2 we can determine the set of urgent actions which will be enabled.

$$F_{\langle q, \alpha \rangle} \quad = \quad \{u_i \in U_q \mid \exists \alpha' \in \mathtt{succ}(\alpha) \cup \{\alpha\}.\ \alpha' \Rightarrow \mathtt{min}(\psi_{u_i})\}$$

If this set is empty, the state $\langle q, \alpha \rangle$ and its outgoing urgent transitions remain unchanged. Otherwise we add to each outgoing urgent transition an explicit constraint which forces it to respect both the expiry time expressed by ℓ and the minimal upper bound of the enabling interval of other urgent transitions. Such an explicit constraint does not modify the behavior of the automaton, but explicitly adds to the transition constraint the conditions expressed by the semantic rules of Section 4.1.

If in $F_{\langle q, \alpha \rangle}$ there are transitions that are already enabled when the state is entered we simply add to each transition in $\bigcup_{i=1}^{k} E_{u_i}^{\alpha}$ a new constraint imposing that the time elapsed in the state be less than ℓ. To do this we add in T_u^r a new clock variable. Whenever a state is entered this clock is reset, so it can be used in the constraints of outgoing edges as a measure of the time elapsed in the state.

If in $F_{\langle q, \alpha \rangle}$ there are only transitions that are enabled after some time, we determine the first clock region which satisfies a constraint of an urgent action $\mathtt{fst\_succ}(\alpha, \mathtt{min}(\psi_{u_i})) = min_{\alpha' \in \mathtt{succ}(\alpha) \cup \{\alpha\}}(\alpha' \Rightarrow \mathtt{min}(\psi_{u_i}) \wedge u_i \in F_{\langle q, \alpha \rangle})$. Note that if $\mathtt{fst\_succ}(\alpha, \mathtt{min}(\psi_{u_i})) = \alpha$ then the urgent transition is already enabled when the state is entered (remember Proposition 4.2). By convention, if the clock constraint $\mathtt{min}(\psi_{u_i})$ is equivalent to false or if it is consistent, but it will never be true letting the time to elapse from α, the result of $\mathtt{fst\_succ}$ is $\top$ and this value has the property of being greater than any element of $\mathtt{succ}(\alpha) \cup \{\alpha\}$. Moreover, we can establish the immediate predecessor, according to the total order, of a clock region α' in the set $\texttt{succ}(\alpha)$. Let us denote this by $\texttt{prec}(\alpha')$. Again, if α' is the minimum in $\texttt{succ}(\alpha)$, then its predecessor is α.

Definition 4.6 (Set of Crucial Clocks) *If $\alpha \not\Rightarrow \texttt{min}(\psi_{u_i})$, then we define the set $\texttt{cruc}(\alpha, \texttt{min}(\psi_{u_i}))$ as the set $\mathcal{X} - \{x \in \mathcal{X} \mid R_{T_u}(\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i}))), x) \Rightarrow \texttt{select}(\texttt{min}(\psi_{u_i}), x)\}$. It contains the only clocks that determine the truth of the constraint $\texttt{min}(\psi_{u_i})$ in the region $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i}))$.*

Example 4.7 *Let us explain the concept of "crucial". Let $\texttt{min}(\psi_{u_i})$ be $0 < x < 2 \wedge 1 < y < 3$. If α is $[x = 0 \wedge 0 < y < 1]$ then $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i})) = [1 < y < 2 \wedge 0 < x < 1, \texttt{fract}(y) < \texttt{fract}(x)]$ and $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i}))) = [y = 1 \wedge 0 < x < 1]$. In $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i})))$, the value of clock x, $0 < x < 1$, implies the atomic constraint $\texttt{select}(\texttt{min}(\psi_{u_i}), x) = 0 < x < 2$, so x is not crucial for $\texttt{min}(\psi_{u_i})$. Instead, the value of y, $y = 1$, does not imply $\texttt{select}(\texttt{min}(\psi_{u_i}), y) = 1 < y < 3$. Thus, we have $y \in \texttt{cruc}(\alpha, \texttt{min}(\psi_{u_i}))$.*

If α is $[y = 1 \wedge x = 0]$ then $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i})) = [1 < y < 2 \wedge 0 < x < 1, \texttt{fract}(y) = \texttt{fract}(x)]$ and $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i})))$ is α itself. Here both x and y are crucial clocks.

Proposition 4.3 *The set of crucial clocks always contains at least one element.*

Proof. For contradiction, suppose it is empty. Then, the constraints of the clock region $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i})))$ would imply $\texttt{min}(\psi_{u_i})$.

But, by definition, $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i}))$ is the minimum clock region that implies $\texttt{min}(\psi_{u_i})$ and $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi_{u_i})))$ is strictly less than it using the order defined in $\texttt{succ}(\alpha) \cup \{\alpha\}$. A contradiction. ■

Let $u_i \in F_{\langle q,\alpha\rangle}$. The constraint $\texttt{select}(\texttt{min}(\psi_{u_i}), x)$, given any crucial clock x, can be used to determine a constraint that force the urgent action to be executed within ℓ time units from the time in which it becomes enabled (and respecting the rules discussed in Section 4.1). To do this we add to any transition in $\bigcup_{i=1}^{k} E_{u_i}^{\alpha}$ the additional constraint $\texttt{add}(\alpha, \texttt{min}(\psi_{u_i})))$ constructed as follows. Given any crucial clock x for $\texttt{min}(\psi_{u_i})$, $x \in \texttt{cruc}(\alpha, \texttt{min}(\psi_{u_i}))$:

- $\texttt{add}(\alpha, \texttt{min}(\psi_{u_i})))$ is $(x < c + \ell)$ if $\texttt{select}(\texttt{min}(\psi_{u_i}), x)$ is either $(x = c)$ or $(c \leq x \# d)$ or $(c \leq x)$, where $c < d$ and $\# \in \{\leq, <\}$.
- $\texttt{add}(\alpha, \texttt{min}(\psi_{u_i})))$ is $(x \leq c + \ell)$ if $\texttt{select}(\texttt{min}(\psi_{u_i}), x)$ is either $(c < x \# d)$ or $(c < x)$ where $c < d$ and $\# \in \{\leq, <\}$.

The final step in the construction of a ℓ consistent timed automaton is to add to all urgent transitions a constraint which forces all of them to be executed (if possible) before the upper bound of the enabling interval of the first disabled one:

recall Figure 4.2(a). If such an urgent transition does not exist, this step, for the current state, ends. To formalize this search, we exploit again the total order defined in $\mathtt{succ}(\alpha)$. Given the set $F_{\langle q,\alpha\rangle}$, we search, for each element $(q, \psi_{u_i}, \gamma_{u_i}, \sigma_{u_i}, q'_{u_i})$ of this set, the clock region, in $\mathtt{succ}(\alpha)$, in which the constraint $\mathtt{min}(\psi_{u_i})$ becomes false, if any. $\mathtt{fst\_dis}(\alpha, \mathtt{min}(\psi_{u_i}))$ denote such clock region. If the constraint will never be false, letting the time to elapse, in the current state, the result of this operation is, by convention, $\top$ and has the property of being greater than any element of $\mathtt{succ}(\alpha)$. Now we can explicitly define the set of enabled urgent transition which are first disabled.

$$\begin{array}{rcl} L_{\langle q,\alpha\rangle} & = & \{u_i \in F_{\langle q,\alpha\rangle} \mid \mathtt{fst\_dis}(\alpha, \mathtt{min}(\psi_{u_i})) \neq \top \;\wedge\; \forall u_j \in F_{\langle q,\alpha\rangle}. \\ & & \qquad \mathtt{fst\_dis}(\alpha, \mathtt{min}(\psi_{u_i})) \leq_\alpha \mathtt{fst\_dis}(\alpha, \mathtt{min}(\psi_{u_j}))\} \end{array}$$

Taking any transition u_i in this set, we have to find the clock constraint to add to other urgent transitions that disables them when u_i is disabled. To this purpose we introduce the following definition.

Definition 4.8 *Let ψ be a constraint without redundancies over a set of clocks $\mathcal{X}$.*

The lower opening $\mathcal{O}^-(\psi)$ *of ψ is obtained by deleting from ψ all the constraints of the form $c \leq x$ and $c < x$, and by substituting all the constraints of the form $x = c$ by $x \leq c$, for all $x \in \mathcal{X}$.*

Analogously, the upper opening $\mathcal{O}^+(\psi)$ *of ψ is obtained by deleting from ψ all the constraints of the form $x \leq c$ and $x < c$, and by substituting all the constraints of the form $x = c$ by $c \leq x$, for all $x \in \mathcal{X}$.*

Clearly this definition requires constraints of the form $c\#x\#d$, $\# \in \{<, \leq\}$, to be considered as $c\#x \wedge x\#d$.

Thus if we want that all the urgent transitions will be disabled when u_i is disabled, we need to add to all of them the constraint $\mathcal{O}^-(\mathtt{min}(\psi_{u_i}))$ which becomes false at the same time of the first disabled urgent transition.

Definition 4.9 *Let $T_u^r = (Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}^r, B^r, R^r, \mathcal{X})$ be a timed automaton with urgent transitions in region form. The* ℓ-consistent *version of it, ℓT_u^r, is the timed automaton $(Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}_\ell^r, B^r, R^r, \mathcal{X}^r)$ where $\mathcal{X}^r = \mathcal{X} \cup \{x_{\mathtt{time\_state}}\}$ and $\mathcal{U}_\ell^r$ is constructed as follows:*

1. $(\langle q,\alpha\rangle, \psi, \gamma \cup \{x_{\mathtt{time\_state}}\}, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}_\ell^r$ *iff* $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$ *and* $F_{\langle q,\alpha\rangle} = \emptyset$

2. $(\langle q,\alpha\rangle, \psi \wedge x_{\mathtt{time\_state}} < \ell, \gamma \cup \{x_{\mathtt{time\_state}}\}, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}_\ell^r$ *iff* $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$ *and* $u \in F_{\langle q,\alpha\rangle}$ *and* $\alpha \Rightarrow \mathtt{min}(\psi_u)$ *and* $L_{\langle q,\alpha\rangle} = \emptyset$

3. $(\langle q,\alpha\rangle, \psi \wedge x_{\mathtt{time\_state}} < \ell \wedge \mathcal{O}^-(\mathtt{min}(\psi_{u'})), \gamma \cup \{x_{\mathtt{time\_state}}\}, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}_\ell^r$ *iff* $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$ *and* $u \in F_{\langle q,\alpha\rangle}$ *and* $\alpha \Rightarrow \mathtt{min}(\psi_u)$ *and* $u' \in L_{\langle q,\alpha\rangle}$

4. $(\langle q,\alpha\rangle, \psi \wedge \mathtt{add}(\alpha, \mathtt{min}(\psi_u)), \gamma \cup \{x_{\mathtt{time\_state}}\}, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r_\ell$ *iff* $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$ *and* $u \in F_{\langle q,\alpha\rangle}$ *and* $\alpha \not\Rightarrow \mathtt{min}(\psi_u)$ *and* $L_{\langle q,\alpha\rangle} = \emptyset$

5. $(\langle q,\alpha\rangle, \psi \wedge \mathtt{add}(\alpha, \mathtt{min}(\psi_u)) \wedge \mathcal{O}^-(\mathtt{min}(\psi_{u'})), \gamma \cup \{x_{\mathtt{time\_state}}\}, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r_\ell$ *iff* $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$ *and* $u \in F_{\langle q,\alpha\rangle}$ *and* $\alpha \not\Rightarrow \mathtt{min}(\psi_u)$ *and* $u' \in L_{\langle q,\alpha\rangle}$

Proposition 4.4 *Given a timed automaton with urgent transitions in region form* T^r_u *and its* ℓ*-consistent version* ℓT^r_u*, then* $\mathcal{L}(T^r_u) = \mathcal{L}(\ell T^r_u)$*.*

Proof. Let us first add the clock $x_{\mathtt{time\_state}}$ to all edges of T^r_u. This operation results in an equivalent timed automaton with urgent transitions.

The proof proceeds for cases.

Case 1. Consider the hypothesis $F_{\langle q,\alpha\rangle} = \emptyset$.

We have

$F_{\langle q,\alpha\rangle} = \emptyset$

$\equiv$ {By definition of $F_{\langle q,\alpha\rangle}$}

$\neg\exists u \in U_q.(\exists\alpha' \in \mathtt{succ}(\alpha) \cup \{\alpha\}.\alpha' \Rightarrow \mathtt{min}(\psi_u))$

$\equiv$ {By Proposition 4.2}

$\neg\exists(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r.[\nu] \in \alpha \cup \mathtt{succ}(\alpha) \wedge \nu \models \psi$

Thus, no urgent action can be performed in $\langle q,\alpha\rangle$ and the state is equivalent in both T^r_u and ℓT^r_u.

Case 2. Consider the hypothesis $u \in F_{\langle q,\alpha\rangle}$ and $\alpha \Rightarrow \mathtt{min}(\psi_u)$ and $L_{\langle q,\alpha\rangle} = \emptyset$.

We have:

(a)

$(u \in F_{\langle q,\alpha\rangle} \wedge \alpha \Rightarrow \mathtt{min}(\psi_u)) \;\wedge\; L_{\langle q,\alpha\rangle} = \emptyset$

$\Rightarrow$ {by definition of $L_{\langle q,\alpha\rangle}$}

$(u \in F_{\langle q,\alpha\rangle} \wedge \alpha \Rightarrow \mathtt{min}(\psi_u)) \wedge$
$(\neg\exists u \in F_{\langle q,\alpha\rangle}. \mid (\exists\alpha' \in \mathtt{succ}(\alpha).\ \mathtt{fst\_dis}(\alpha, \mathtt{min}(\psi_u)) \neq \top))$

$\Rightarrow$ {by definition of $F_{\langle q,\alpha\rangle}$, $L_{\langle q,\alpha\rangle}$ and Proposition 4.2}

$\forall\nu : [\nu] \in \alpha \cup \mathtt{succ}(\alpha).$
$(\forall\delta_{\langle q,\alpha\rangle} \geq \ell.(\exists u = \langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r.(\exists\delta.0 \leq \delta \leq \delta_{\langle q,\alpha\rangle} \wedge \delta \geq \ell \wedge \nu - \delta \models \psi)) \wedge$
$(\forall\delta_{\langle q,\alpha\rangle} < \ell.(\neg\exists u = \langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r.(\exists\delta.0 \leq \delta \leq \delta_{\langle q,\alpha\rangle} \wedge \delta \geq \ell \wedge \nu - \delta \models \psi)) \wedge$

$\forall \delta_{\langle q,\alpha\rangle}.(\neg\exists u = \langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r.(\exists \delta.0 \leq \delta \leq \delta_{\langle q,\alpha\rangle} \wedge \nu \not\models \psi \wedge \nu - \delta \models \psi)$

Now, let us consider $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$.

The condition for performing such an action, given by the **(Urgent)** rule is:

$\nu \models \psi \wedge$
$(\neg\exists u' = (\langle q,\alpha\rangle, \psi_{u'}, \gamma_{u'}, \sigma_{u'}, \langle q',\alpha'\rangle) \in \mathcal{U}^r.$
$(\exists \delta.0 \leq \delta \leq \delta_{\langle q,\alpha\rangle} \wedge \delta \geq \ell \wedge \nu - \delta \models \psi_{u'}))$
$(\neg\exists u' = (\langle q,\alpha\rangle, \psi_{u'}, \gamma_{u'}, \sigma_{u'}, \langle q',\alpha'\rangle) \in \mathcal{U}^r.$
$(\exists \delta.0 \leq \delta \leq \delta_{\langle q,\alpha\rangle} \wedge \nu \not\models \psi_{u'} \wedge \nu - \delta \models \psi_{u'}))$

which, under the hypothesis and proof (a), is equivalent to:

$\nu \models \psi \wedge \delta_{\langle q,\alpha\rangle} < \ell$

Now $\nu \models \psi \wedge \delta_{\langle q,\alpha\rangle} < \ell$ iff $\nu \models (\psi \wedge x_{\texttt{time\_state}} < \ell)$, thus an urgent action $(\langle q,\alpha\rangle, \psi \wedge x_{\texttt{time\_state}} < \ell, \gamma \cup \{x_{\texttt{time\_state}}\}, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r_\ell$ can be performed if and only if $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$ can be performed.

Case 3. Consider the hypothesis $u \in F_{\langle q,\alpha\rangle}$ and $\alpha \Rightarrow \texttt{min}(\psi_u)$ and $u' \in L_{\langle q,\alpha\rangle}$.

Under the hypothesis $u \in F_{\langle q,\alpha\rangle}$ and $\alpha \Rightarrow \texttt{min}(\psi_u)$, the proof of Case 2. states that the introduction of the constraint $x_{\texttt{time\_state}} < \ell$ guarantees that no urgent action can be performed after ℓ time units from the enabling of the first urgent action.

Now we have an additional hypothesis:

$u' \in L_{\langle q,\alpha\rangle}$

$\Rightarrow$ {By the definitions of $L_{\langle q,\alpha\rangle}$, $\mathcal{O}^-$ and the hypothesis}

$\forall \nu : [\nu] \in \alpha \cup \texttt{succ}(\alpha).\nu \not\models \mathcal{O}^-(\texttt{min}(\psi_{u'})) \equiv ((\exists \delta.0 \leq \delta \leq \delta_{\langle q,\alpha\rangle} \wedge \nu \not\models \psi_{u'} \wedge \nu - \delta \models \psi_{u'}))$

Thus, for the **(Urgent)** rule, an urgent action could be performed only if $\nu \models \mathcal{O}^-(\texttt{min}(\psi_{u'}))$. We can conclude that an urgent action $(\langle q,\alpha\rangle, \psi \wedge x_{\texttt{time\_state}} < \ell \wedge \mathcal{O}^-(\texttt{min}(\psi_{u'}), \gamma \cup \{x_{\texttt{time\_state}}\}, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r_\ell$ can be performed if and only if $(\langle q,\alpha\rangle, \psi, \gamma, \sigma, \langle q',\alpha'\rangle) \in \mathcal{U}^r$ can be performed.

Cases 4. and 5. The proof proceeds analogously to the ones of the other cases.

■

4.2.3 The quiet version of a timed automaton with urgent transitions

Now, in order to achieve the desired behavior, in each state of ℓT_u^r we have to turn off all the originally non-urgent outgoing transitions when at least one of the edges obtained by the originally urgent transition is enabled. This is the third transformation step.

Let $e = (\langle q, \alpha\rangle, \psi, \gamma, \sigma, \langle q', \alpha'\rangle)$ be a non-urgent outgoing transition from a state in ℓT_u^r. We map e in some transitions $e' = (\langle q, \alpha\rangle, \psi \wedge \theta, \gamma, \sigma, \langle q', \alpha'\rangle)$ of the new automaton where θ is the constraint that will become false when at least one of the outgoing urgent transitions of the state $\langle q, \alpha\rangle$ in ℓT_u^r becomes true. The disjunction of the upper opening (Definition 4.8) of all urgent edges constraints (without redundancies) outgoing from a state in ℓT_u^r, describes a right-infinite time interval to the beginning of which an urgent transition must be taken. The negation of this disjunction must be added to all the constraints of non-urgent edges of T_u^r outgoing from the same state.

The negation of a complex formula can introduce disjunction of constraints. We denote by $\mathtt{DNF}^+$ an operation that, given a constraint which contains negations, push the negation operator inside, using the logical axioms for $\neg, \wedge, \vee$, until it is applied to atomic constraints. Then it transforms the negations of these constraints to the correspondent positive ones ($x = c$ will be translated into $x < c \vee c < x$). Finally, it transforms the formula in disjunctive normal form. It returns the set containing all the conjunctive components of the formula.

Definition 4.10 *Let $\ell T_u^r = (Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}_\ell^r, B^r, R^r, \mathcal{X}^r)$ be the ℓ-consistent version of a timed automaton with urgent transitions in region form T_u^r. The* quiet *version of it, T_u^{quiet}, is the timed automaton $(Q^r, \Sigma, \mathcal{E} = \mathcal{U}_\ell^r \cup \mathcal{E}', B^r, R^r, \mathcal{X}^r)$ where $\mathcal{E}'$ is constructed as follows:*
$(\langle q, \alpha\rangle, \psi \wedge \phi, \gamma \cup \{x_{\langle q', \alpha'\rangle}\}, \sigma, \langle q', \alpha'\rangle) \in \mathcal{E}'$ iff $(\langle q, \alpha\rangle, \psi, \gamma, \sigma, \langle q', \alpha'\rangle) \in \mathcal{E}^r$, and $\phi \in \mathtt{DNF}^+(\neg(\bigvee_u \mathcal{O}^+(\mathtt{min}(\psi_u)))$ for all $(\langle q, \alpha\rangle, \psi_u, \gamma_u, \sigma_u, \langle q'', \alpha''\rangle) \in \mathcal{U}_\ell^r$.

Example 4.11 *Figure 4.6 shows the automaton T^{1quiet} which is the 1-consistent and quieted version of the automaton T_u^1 of Figure 4.4. Note that the constraint $1 < x$ on one of the edges for b has been modified to $1 < x \wedge x \leq 1$ by the last transformation. Thus, being always false has been removed. In figure, the clock $x_{\mathtt{time\_state}}$ is omitted because it is useless in this case.*

Theorem 4.5 *Let T_u be a timed automaton with urgent transitions, T_u^r the corresponding timed automata in region form and T_u^{quiet} its quiet version. Then $\mathcal{L}(T_u) = \mathcal{L}(T^{quiet})$.*

Proof. Starting from the result of Proposition 4.4, the proof proceeds analogously to the one of Proposition 4.4 itself. ■

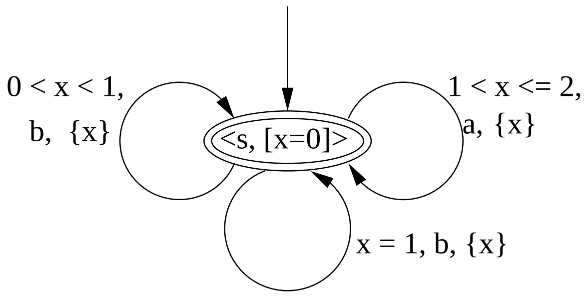

Figure 4.6: Automaton T^{1quiet}

As a last remark of this section we want to discuss the size of the output of the transformation. Indeed, it turns out that our notion of urgent actions is a succinct way to express urgency. To see this consider that one could just design his/her automaton as the quiet version we obtain by the transformation. That is to say, one can think about states as pairs $\langle q, \alpha \rangle$ and use suitable constraints to express urgency as we do by the algorithm.

However, it is clear that the quiet automaton has a larger size than the automaton with urgent transitions. As a first approximation the size of the quiet version of the automaton is at least the size of the region automaton (see Section 2.2.1). Here we consider the size of a timed automaton as the sum of the number of states and the number of clock constraints on the edges. The size of the quiet version is greater than that because we add constraints (and edges) to induce the behavior specified by the formal semantics.

4.3 Specification using urgent transitions

In the previous section we defined a new feature for the timed automata specification formalism. After that we showed how to compile a timed automaton with urgent transition into a standard timed automaton.

Indeed, a specification formalism needs a way to define systems as a composition of components. For timed automata this mechanism is the parallel composition (see Section 1.4.2). The parallel composition of timed automata with urgent transitions is defined in the same way, but observing the following remarks:

- the urgency of a transition of a component with a synchronization action σ extends to the transition obtained using one of the rules in part 1 of Definition 1.12,

- the urgency of a transition of a component extends to the transition obtained by the rules in parts 2 and 3 of Definition 1.12.

It is easy to see that the transformation defined in the previous section *is not a congruence with respect to parallel composition*. In other words if we have two timed automata, T_u^1 and T_u^2, with urgent transition the automaton $T_u^1 \parallel T_u^2$, when defined, is not equivalent, in general, to $T_1^{quiet} \parallel T_2^{quiet}$ (the standard parallel composition of the quiet version of them).

Figure 4.7: Automaton A_u^1 and its quiet version $A^{1\,quiet}$

Example 4.12 *Consider the automaton A_u^1 of Figure 4.7(a), the automaton T_u^1 of Figure 4.4 and the parallel composition $T_u^1 \parallel A_u^1$. The action c, having the constraint $y > 0$ and being urgent, surely preempts the action b at the beginning of every run. The parallel composition of the quiet version of T_u^1 (Figure 4.6) and of the quiet version of A_u^1 (Figure 4.7(b)) has a different behavior: in the automaton $T^{1\,quiet} \parallel A^{1\,quiet}$ the action b can be performed at the beginning of a run before the urgent action c. This is because the transformation of T_u^1 into $T^{1\,quiet}$ does not consider the urgency of the action c belonging to the component $A^{1\,quiet}$.*

Thus, the interpretation of the non-urgent outgoing transitions in a state of an automaton with respect to the urgency of a transition depends on the context in which the automaton is considered. If it is viewed as a component instead of a stand-alone system, the enabling of those transitions should change taking into account other component's urgent transitions that could interleave with them. This aspect turns out explicitly in the example of the following section.

4.4 An example

As an example of specification of a system with urgent actions we use a part of a multicast protocol for mobile computing presented in [ABDS00]. There the protocol is specified by the Calculus of Communicating Systems (CCS) [Mil80], and the Concurrency Workbench tool [CS96] is used to state some required properties.

Let us start with an informal description of the protocol, quoted from [ABDS00]:

"...We consider a system composed of mobile hosts and stationary hosts.

Communication occurs solely via message-passing. Some stationary hosts (gateways) are connected to a wired network that provides reliable and FIFO-ordered communication and to a wireless link, that covers a spatially limited *cell* nearby the gateway. Mobile hosts may move and communicate through wireless links. A gateway may broadcast messages to all mobile hosts in its cell. A mobile host may only exchange messages with the gateway of the cell where it happens to be located.

We shall associate a different meaning with the terms “receiving” and “delivering” a message. A host C *receives* a message msg when msg arrives at its protocol layer. Upon receiving msg, the protocol may discard msg or pass msg up to one of its applications. In the latter case, C is said to *deliver* msg.

The protocol works as follows. A dedicated stationary host acts as the *coordinator*, denoted as SC. A mobile host generates a multicast by sending a message to a gateway, which forward it to SC. SC constructs a message containing an increasing sequence number, then transmits the resulting message to gateways through a FIFO-multicast. Gateways broadcast this message to mobile hosts in the respective cells.

Due to its movement across cells, any mobile host m could receive duplicates or could miss multicasts. By maintaining a history of the received sequence numbers, m discards duplicates in the former case and sends to the stationary hosts a proper *nack* message in the latter (e.g., upon receiving an out-of-order message). Upon receiving a *nack*, the stationary host will relay to m a copy of the missing multicasts.”

The protocol was designed to guarantee several properties, in particular: ”Each mobile host m delivers each multicast under reasonable assumptions, as follows: m stops delivering messages if: (i) m starts entering and leaving cells so quickly that its messages never arrive to any stationary host or messages from gateways are systematically lost; and (ii) this pattern of movements persists forever.

. . . ”

The explicit assumption in the formulation of the property is due to the fact that CCS specifications cannot express constraints on the speed of mobile hosts. Thus, using this specifications, the system can have “bad” behaviors, which must be ruled out in the property formulation. This means that the verification of the property is restricted to all the possible “good” behaviors. That is, the execution paths in which the mobile host enters and leaves cells without getting any message, although possible, are disregarded.

With timed automata with urgent transitions we can easily describe the behavior of the components involved in the protocol together with important time parameters such as the speed of a mobile agent, that is the minimal amount of time it can stay in a cell or the frequency it exchanges messages with the stationary hosts. This allows to state the property on the whole system behavior which, due to time parameters, should not allow “bad” executions.

To make a simple example of this feature of timed automata with urgent transitions, we extract from the above presented protocol the part dealing with the iterated request of a mobile host to broadcast a message until it effectively receives it. For reasons of simplicity we assume only one mobile host, two stationary ones and the coordinator, as shown in Figure 4.8. Of course, the mobile host, being unique, cannot receive multicasts generated by other mobile hosts. We assume that multicasts are generated directly by the gateways.

The system is specified by the parallel composition of the automata specifying each entity.

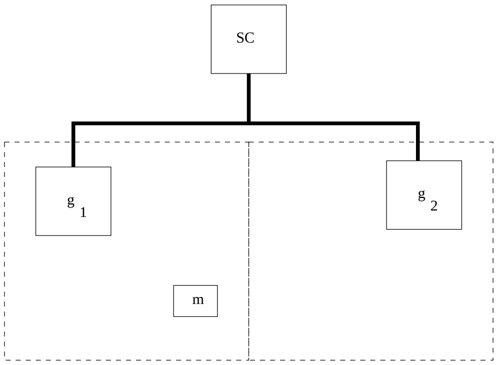

Figure 4.8: The scenario

The coordinator SC is defined in Figure 4.9. It accepts a request from a gateway g_j ($\mathtt{req}_{g_j}(i)$) for broadcasting the $i-th$ message. Then it tries to contact all the gateways to signal that the $i-th$ message must be sent to their cells. Note that the coordinator tries to contact the stationary hosts sequentially, it passes to contact the next one only if either the previous accepted the request ($\mathtt{signal}_{g_j}(i)$) or it does not respond for twenty time units ($\mathtt{fail}_{g_j}$). Note that in the latter case the coordinator proceeds in, at most, twenty one time units. It is important to remark that the action of contacting the stationary hosts is urgent. That is, when enabled, it cannot be skipped and it must be done within a given time interval. One could observe that the automaton always have the urgent transition enabled when it enters state 2 (and 3) and so the $\mathtt{fail}_{g_1}(i)$ ($\mathtt{fail}_{g_2}(i)$) transition can never be taken. But, as observed in Section 4.3, this automaton has to be understood as a component of a whole system. This means that the semantics of urgency is defined in terms of the whole system where the action $\mathtt{signal}_{g_1}(i)$ (and $\mathtt{signal}_{g_1}(i)$) is a synchronization action and can be executed only if the partner (the gateway) can execute it. This depends on the state in which it currently is. Thus, in some runs of the whole system, it could happen that the action $\mathtt{fail}_{g_1}(i)$ ($\mathtt{fail}_{g_2}(i)$) is taken depending on the relative speed of the components and on the interleaving of non-synchronization actions. Note that without urgent transitions we cannot impose the priority and the urgency of the action $\mathtt{signal}_{g_1}(i)$ ($\mathtt{signal}_{g_1}(i)$) with respect to the action $\mathtt{fail}_{g_1}(i)$ ($\mathtt{fail}_{g_2}(i)$) in state 2 (3) because the standard semantics of timed automata would execute them non-deterministically.

A gateway g_j is defined in Figure 4.10. It passes the requests for broadcasting the $i-th$ message from the mobile host ($\mathtt{reqm}_{g_j}(i)$) to the coordinator ($\mathtt{req}_{g_j}(i)$), and broadcasts the message $\mathtt{msg}_{g_j}(i)$ on its cell upon request of the coordinator ($\mathtt{signal}_{g_j}(i)$). Note that the action of sending the message is urgent, it can be skipped only if the mobile host do not respond within ten time units. The same remarks on urgency on parallel composition given above applies here.

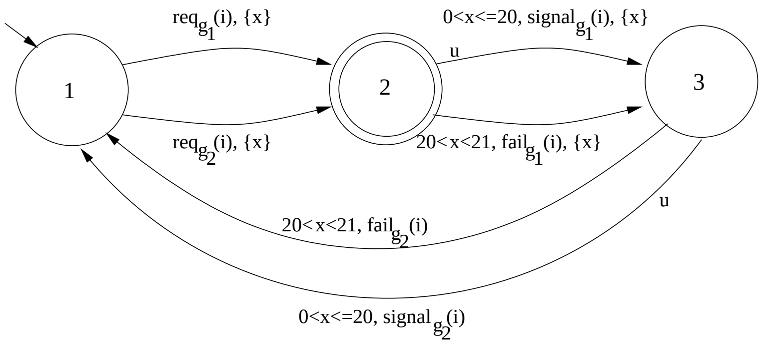

Figure 4.9: The coordinator SC

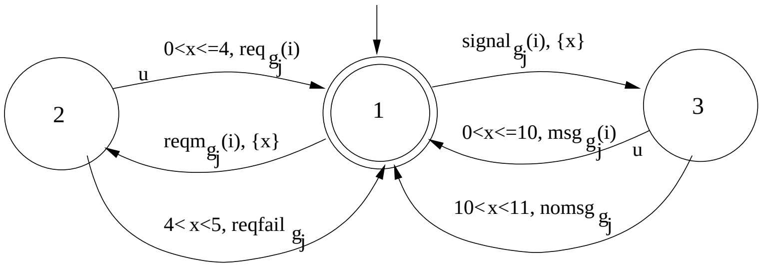

Figure 4.10: The gateway g_j

Finally, the mobile host m is defined in Figure 4.11. It tries every four units of time to contact the gateway for requesting to broadcast the $i-th$ message for a duration of one time unit. When the $i-th$ message is received and delivered a message counter is increased. Note that, due to parallel composition, both actions are urgent, so they must be done if enabled. Moreover, the mobile host can move from a cell to another; however its speed cannot allow to stay in a cell less than fifty time units. This action is not urgent, that is a host can stay in a cell also if the action of moving is enabled.

Our notion of urgency expresses both a priority among actions and a time constraint on their execution. Both these concepts are useful in specifying real systems. In the scenario of Figure 4.8, the mobile host, when staying within a cell, must first of all try to communicate with the gateway and, if possible, it must do it immediately. If we remove the urgency annotation from the previous automata, the mobile host could never communicate while waiting for the time of moving.

With this specification, the correctness property of the protocol can be simply restated as: "Each mobile host m delivers each multicast".

Because we do not yet have a tool implementing our notion of urgency, we have used the features of UPPAAL [BLL$^+$96] to verify the protocol. In particular we used its notions of urgent channels to simulate our notion of urgency. These features are not sufficient to express the notion that we have introduced in this chapter and thus

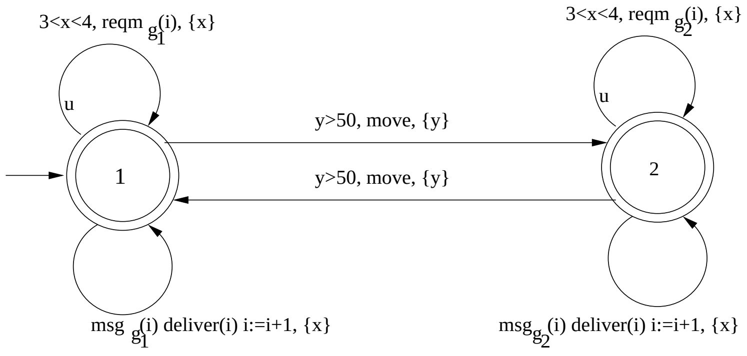

Figure 4.11: The mobile host m

the system that we have tested is an approximation of the one shown in this section. See Section 4.7 for the details.

4.5 Notes on Implementation

In this section we address two problems that arise in the effective use of timed automata with urgent transitions and the defined transformation. The first is the progress model used by the automata and the second is the state explosion due to the region automaton construction.

We have used, to introduce urgent transitions, the model of timed Büchi automata (Chapter 1). In Section 2.3 we have discussed the problem in the implementation of this model and we have introduced timed safety automata (Section 2.3.1) as implementable timed automata. It is easy to see that the definition of timed automata with urgent transitions and the transformation that we have developed is the same if the model of timed safety automata is used. The only difference is in the construction of the region form of the automaton in which every state $\langle s, \alpha \rangle$ has the invariant condition of the state s in the original automaton. The proof of correctness is the same.

The problem of state explosion in verification using timed automata is a serious aspect that has been attacked from the beginning of the presentation of the model. The main source of this explosion is the region construction that we have recalled in Section 2.2.1. The number of regions is exponential in the number of clocks and in the magnitude of the largest integer constant used in the clock constraints. The main tool to contain this complexity is the use of clock *zones* instead of clock regions. This aspect has been treated in Section 2.2.3.

For the sake of simplicity and clarity of exposition we have used, in our transformation of Section 4.2, the region form of a timed automaton (Definition 4.4). Such a device is very useful to define and, for its properties, also to prove the correctness of the transformation. However, it suffers of the problem of state explosion.

With regard to this, we state that it is possible to define a transformation that uses clock zones instead of clock regions. Such a transformation starts with the definition of the *zone form* of a timed automaton. This automaton is the analogous of the region form of the timed automaton if the zone automaton construction [ACH+92, Alu99, Yov96] is used instead of the region automaton construction. The second and the third step of the transformation remain unchanged.

4.6 Related works

The notion of urgency and/or priority for timed formalisms has been studied in the past. In [BL92] the urgency of actions has been investigated in the process algebra field with the concept of discrete time. Different notions of priority have been introduced for timed automata in [FPY02] and for timed process algebras in [BAL02, BL97].

A closer approach to ours can be found in [BS97, BST98, BS00]. There the states of a timed automaton are associated with time progress conditions (TPC). TPC are state conditions which specify that the time can progress at a state by δ only if all the intermediate times δ', $0 \leq \delta' < \delta$, satisfy it.

TPC are computed from *deadlines*. Deadlines are clock constraints associated to transitions in addition to the usual constraints (which, in this setting, are called *guards*). The defined class of timed automata is called *Timed Automata with Deadlines* (TAD).

Given a state q, its TPC is intuitively computed as follows. Consider the set $I = \{i \mid t_i \text{ is a transition outgoing from } q\}$ of indexes of transitions from q. The TPC of q, c_q, is obtained as the negation of the disjunction of the deadlines, d_i, of all the transitions from q, $c_q = \neg \bigvee_{i \in I} d_i$. In a state of a run, (q, ν), the time can progress by δ, $(q, \nu) \xrightarrow{\delta} (q, \nu + \delta)$, if $\forall \delta' < \delta . \nu + \delta' \models c_q$.

Given a transition in a TAD, with guard ψ and deadline d, we can find in [BST98] the following remark.

"The relative position of d with respect to ψ determines the urgency of the action. For a given ψ, the corresponding d may take two extreme values: first, $d = \psi$, meaning that the action is eager and, second, $d = false$, meaning that the action is lazy. A particularly interesting case is the one of a delayable action *where d is the falling edge of a right-closed guard ψ (cannot be disabled without enforcing its execution).*

The condition $d \Rightarrow \psi$ guarantees that if time cannot progress at some state, then at least one action is enabled from this state. Restriction to right-open TPC guarantees that deadlines can be reached by continuous time trajectories and permits to avoid deadlock situations in the case of eager transitions. For instance, consider the case where $d = \psi = x > 2$, implying the TPC $x \leq 2$, which is not right-open. Then, if x is initially 2, time cannot progress by any delay ψ, according to above definition. The guard ψ is not satisfied either, thus, the system is deadlocked."

This limitation is very intuitive: if the eager transition has a left-open guard, the time at which it can be fired is undefined. Using our concept of urgent transition we avoid this problem because the transition can be fired in the interval in which x is in $[2, 2+\ell)$. On the other hand, if a transition with a left-closed guard, say $x \geq 2$, needs to be fired "as soon as possible", then we can approximate this behavior using a constant ℓ small as needed.

With regard to the problem of right-closed TPCs it is also interesting to note that the introduction of urgent transitions in the tools of verification of timed automata is a difficult task. By now, the latest version of UPPAAL[2] has introduced the concept of "urgent channels"[3] which model transitions that must be fired as soon as possible. But their use is very restricting as we show in the following section.

4.7 Analysis with UPPAAL

We have tried to use the features of the tool UPPAAL to model the part of the multicast protocol introduced in Section 4.4. The system that we present in this Section is an approximation of the one modeled by timed automata with urgent transitions because the notion of urgency supported by UPPAAL is weaker than the one we have proposed.

In UPPAAL there exists the possibility to declare a synchronization channel between two timed processes as "urgent". This kind of urgent channel cannot be constrained by clock constraints. Only conditions on the value of variables can be put as guards of the urgent channel. Whenever two timed processes connected by the urgent channel are allowed to perform the synchronization, this must occur without letting any time elapses.

Note that variables in UPPAAL can change their values only when a non-time transition occurs and an assignment is performed. Thus, when the parallel composition of some timed processes in UPPAAL enters a new state after a non-time transition, the boolean value of a guards on variables is known and it remains unchanged until another non-time transition is performed.

In this way, the main problem with urgent transitions, i.e. the fact that they can become enabled in a state as a real-time elapses and at a certain point they possibly become disabled is avoided and the implementation of urgent channels in UPPAAL is simple.

Our notion of urgency and that one of UPPAAL coincide only when the urgent transition has a clock constraint equivalent to $true$ or when it is known that it is always enabled when its source state is entered and its clock constraint is an upper bound of the values of the clocks.

In any case, however, the concept of the interval (bounded by ℓ) in which an enabled urgent transition must be taken cannot be modeled in any way with the

[2]UPPAAL Version 3.4.0 downloadable at `http://www.uppaal.com`

[3]See the documentation at `http://www.uppaal.com`

current features of UPPAAL.

Figure 4.12 shows the timed process `Coordinator`. The picture has been taken by UPPAAL and the graphical conventions are slightly different from the ones used throughout this thesis. In particular, the double-circled states are initial states and, as outlined in Section 2.4.2, states has invariants and only synchronizations transitions (channels) have labels. In this example all synchronization channels are urgent. State $c0$ acts as the state 0 of the coordinator in Figure 4.9. It simply accepts a request of broadcasting message 1 or 2 from Gateway 1 or 2 through the urgent channels $\mathtt{regg}\,j\,i$ where j is the number of the gateway and i is the number of the message.

For brevity we have modeled only the delivering of 2 messages, but it is clear that we can model the system for any fixed number n of messages. This also is true for the number of gateways and mobile units. In state $c1$ (or equivalently $c8$ for message 2) of the Coordinator there is the first simulation of the behavior of the automaton in Figure 4.9. The urgent transition labeled $\mathtt{signal}_{g_1}(i)$ and having the clock constraint $0 < x \leq 20$ is simulated by the state invariant $x \leq 20$ in the state $c1$ and by two outgoing transitions. One is on the urgent channel $\mathtt{signal}\,1\,1$ and the other, with constraint $x \geq 20$, correspond to the $\mathtt{fail}_{g_1}(1)$ of the original coordinator. If Gateway 1 is enabled to perform the synchronization while the control is in $c1$ the transition is taken immediately and state $c2$ is reached. In this state we simulate the interval with length $\ell = 1$, associated to the urgent transition, by the invariant $y < 1$. If the Gateway 1 is not enabled to perform the synchronization on channel $\mathtt{signal}\,1\,1$ for all the time that the coordinator is in state $c1$, then there is a failure. In every case state $c4$, which corresponds to state 3 of the original automaton, is reached. The other paths on the automaton uses the same technique changing the message number or the gateway number.

The gateways are represented in Figures 4.13 and 4.14. Consider Gateway 1. State $g0$ corresponds to the initial state 1 of the original gateway of Figure 4.10. If the gateway receive a message from the coordinator ($\mathtt{signal}\,1\,i$) then it goes on state $g4$ or $g8$ depending on the number of the message. In every case, it tries to send message i to the mobile unit trough the urgent channels $\mathtt{msg}\,1\,i$ using the same technique that we have seen above to simulate our notion of urgency.

If the gateway receives a request from the mobile unit for a message ($\mathtt{reqmg}\,1\,i$) then it tries to transmit the request to the coordinator through the urgent channel $\mathtt{regg}\,1\,i$ using the same technique. The other behaviors are similar.

Finally, the mobile unit is showed in Figure 4.15. The original one was the automaton in Figure 4.11. In this case, the urgent transition from the initial state, labeled $\mathtt{reqm}_{g_1}(i)$, cannot be simulated as in the other cases because its clock constraint is $3 < x < 4$. This means that when the state is entered the constraint of the urgent transition could not yet be true and it can become true after the elapsing of some time. We cannot simulate the behavior of urgent transitions in this case and. We approximate enabling the urgent transition immediately when the state is entered. Thus, in state $m0$ there are two urgent synchronizations in parallel and the

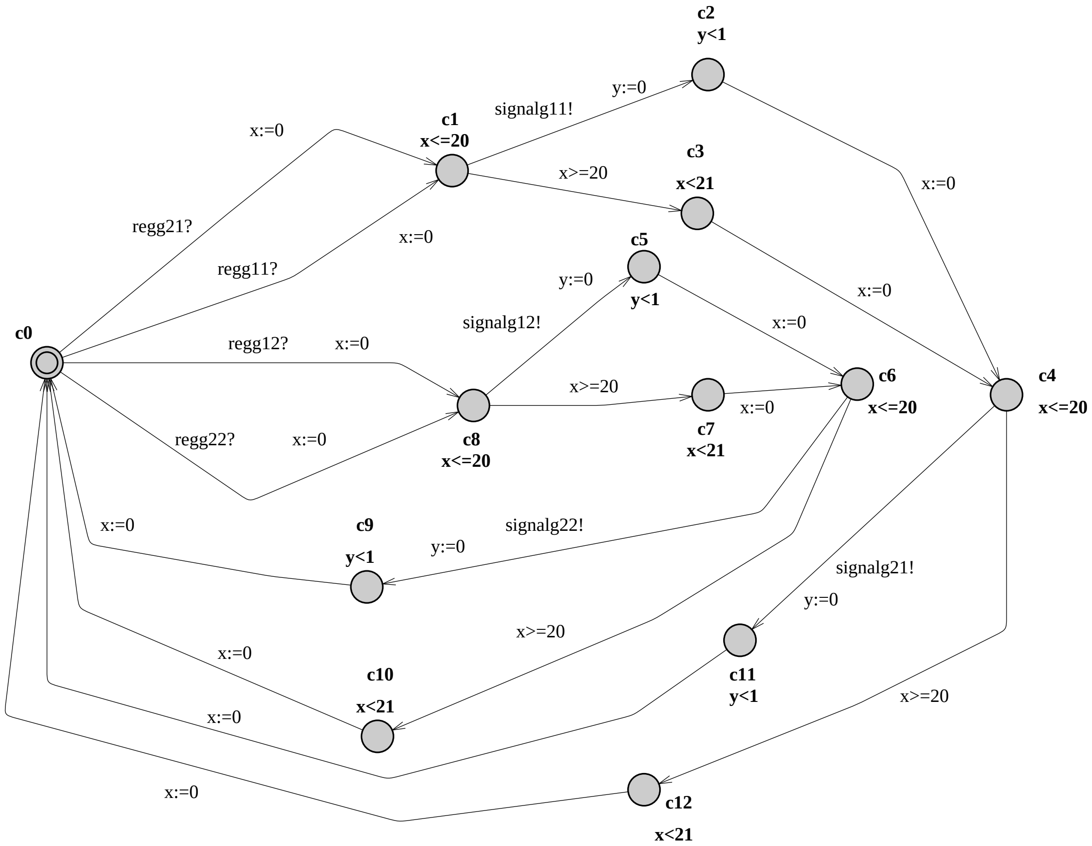

Figure 4.12: The timed process `Coordinator`

possibility for the mobile unit to move on the other cell after 50 time units. States $m0$ corresponds to state 1 of the original automaton. In state $m0$ the message 1 is requested and the automaton attempts to receive it back from the gateway. State $m2$ is symmetric to $m0$ and records the fact that the mobile unit has moved to the cell of Gateway 2. States $m7$ and $m8$ attempt to request and receive back message 2. State $m12$ is the state in which the mobile unit has received all the two messages.

The correctness property can be expressed by the formula

$$\forall\Diamond\ m12$$

i.e. for all paths, eventually the state $m12$ of the mobile unit is reached. The truth of this formula with respect to the given system has been verified by UPPAAL. It is important to remark that, as we said in Section 4.4, if we increase enough the speed of the mobile the property is not satisfied any more. With the times used in this specific example the property becomes false if we put the constraint $w > 1$ instead of $w > 50$ on the transitions that represent the moving of the mobile unit among cells.

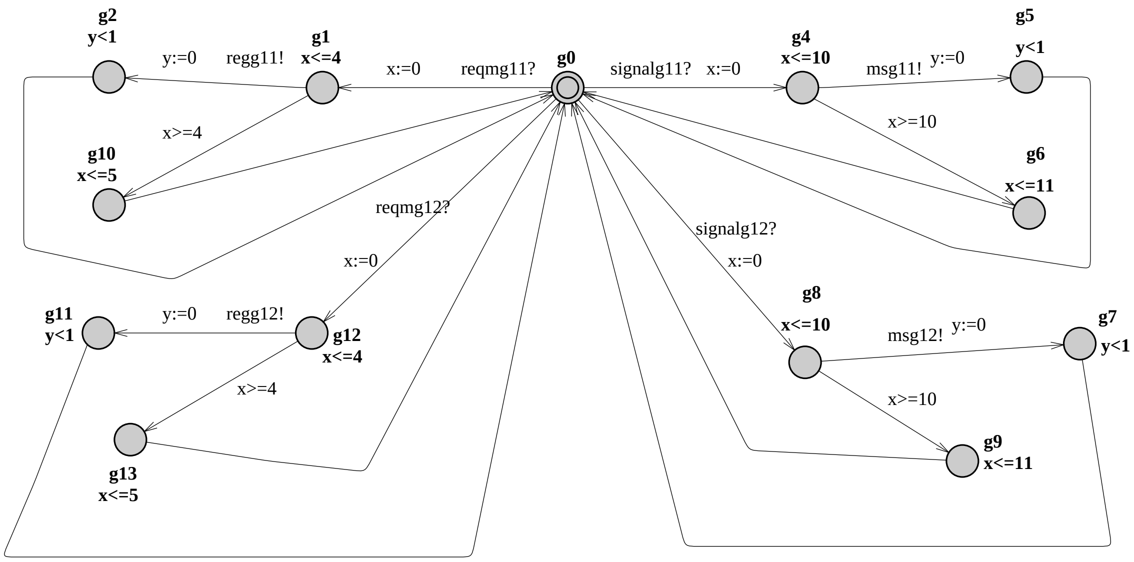

Figure 4.13: The timed process `Gateway1`

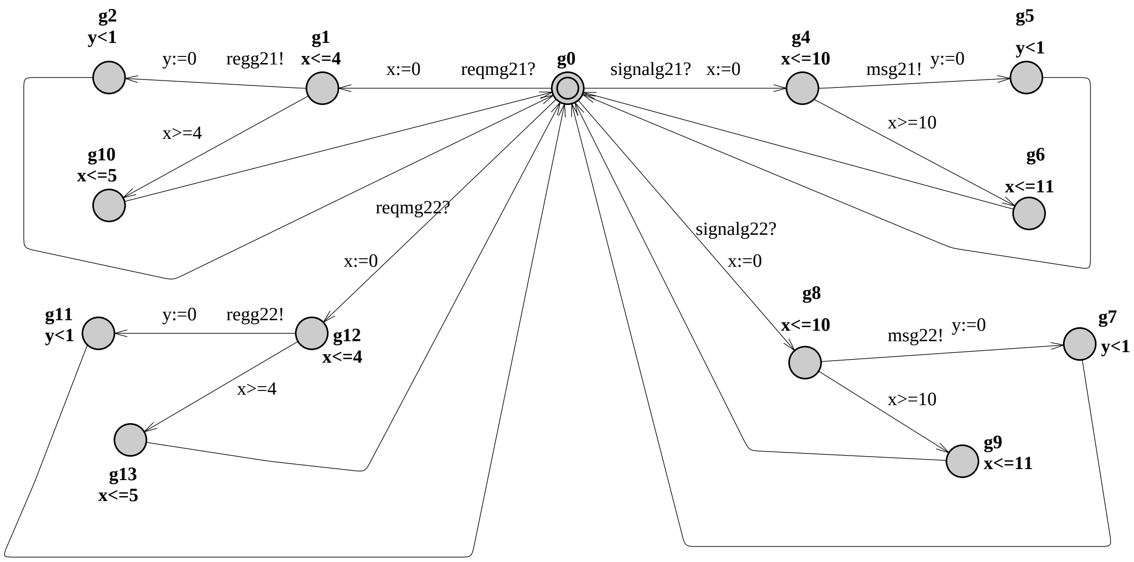

Figure 4.14: The timed process `Gateway2`

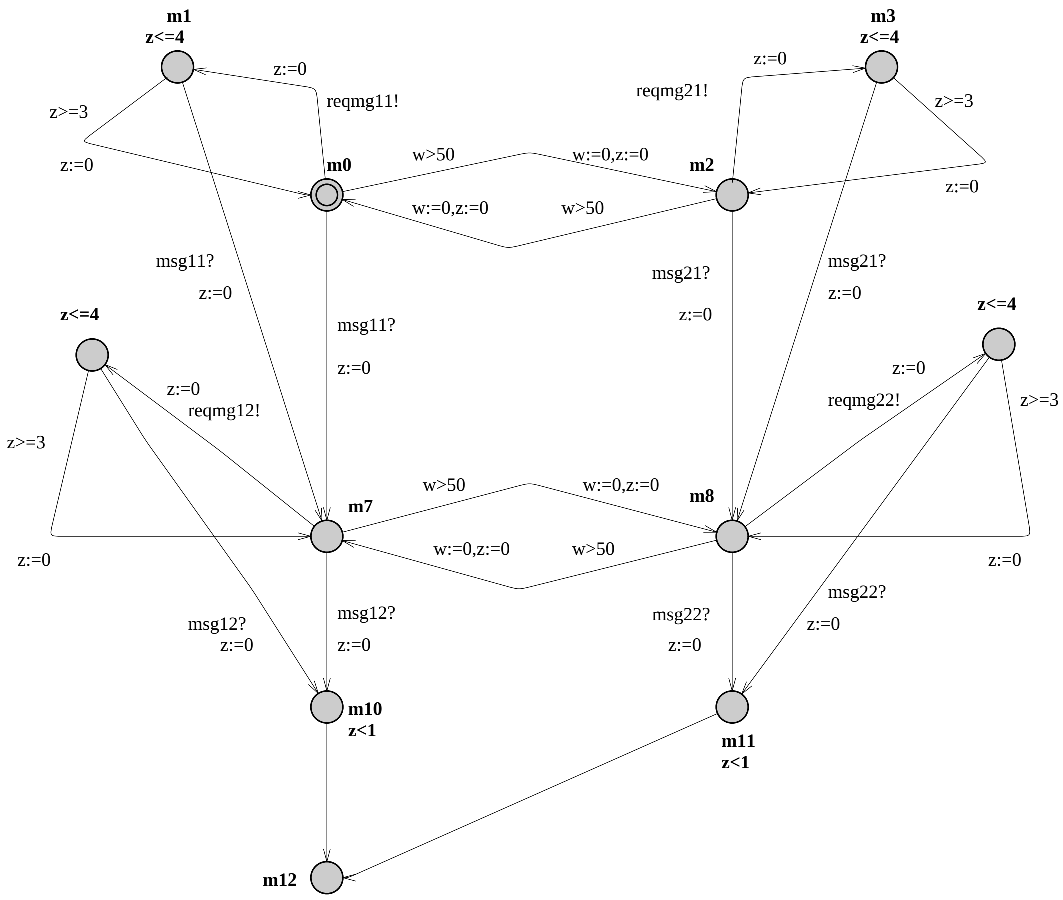

Figure 4.15: The timed process `Mobile`

The diagnostic trace given by UPPAAL corresponds to the situation that is ruled out in the original formulation of the correctness property, that is to say, the mobile unit starts an infinite cycle in which enters and leaves cells without receiving any message. Thus, the speed of the mobile unit is a crucial parameter for determining the truth of the property.

Chapter 5

A Notion of Timed non-Interference

Abstract

In this chapter we contribute to the framework of the verification of real-time systems introducing in it a new idea that has been fruitful in the area of untimed concurrent systems. The new definition is based on existing models and on existing verification techniques. The non-interference property of concurrent systems is a security property concerning the flow of information among different levels of security of a system. It has been used as a basis to define several security properties of multi-level security systems and of several security properties of communication protocols. We introduce a notion of timed non-interference for real-time systems specified by timed automata. The notion is presented using an automata-based approach and then it is characterized also by operations and equivalence between timed languages. The first formulation of the timed non-interference is undecidable. It is applied to an example of a time-critical system modeling a simplified control of an airplane. The presented notion can also be used to verify the strength of a system against attacks depending on the frequency of certain actions. In particular, we give an alternative formulation of timed non-interference which is decidable and can be checked using existing verification tools. We show an application example of this decidable notion by defining a variant of the classical Fischer's mutual exclusion protocol and by analyzing its strength against attacks. The tool UPPAAL is used to perform the automatic verification.

The initial notion of timed non-interference was originally presented in [BDST01] and then the contribution has been published as [BDST02]. The decidable variant of timed non-interference was presented in [BT02] and the full paper has been published as [BT03].

Non-interference is a security property for multilevel security systems. This property has been formulated within different formalisms. In Section 5.1 we recall the general idea. In the framework of real-time systems the notion has not been defined although there can be several fields of applications. As an example, a class of possible attacks to network components is based on use of the time. Time can be used to gather information, as in [DKL+98, Koc96, Sch00], where the execution time is used to reveal a secret key. Time can be also used in direct attacks. If the frequency of the intrusion is high enough, the attacked system cannot work properly.

We define a timed notion of non-interference using timed automata and timed languages in Section 5.2. It is suitable to detect interference due to the frequency of certain actions.

The first formulation suffers of a negative result of undecidability due to the fact that the equivalence between timed automata is used in its definition. For this reason in Section 5.4 we define a decidable notion of timed non-interference that is derived from the first one.

To show a possible application of the decidable timed non-interference we present, in Sections 5.5 and 5.6, an analysis of the strength of a variant of the well-known Fischer's mutual exclusion protocol against an attack which depends on time. We show that we can individuate a certain frequency for the actions of the attacker under which the protocol is not breakable.

Finally, in Section 5.7 we model the example system with the tool UPPAAL and we perform the non-interference verification automatically.

5.1 The Idea of non-Interference from Concurrent Systems

The non-interference property of concurrent systems is a security property, introduced in [GM82], concerning the flow of information among different levels of security of the system. Suppose, for simplicity, that the security levels are two: *high* and *low*. Thus, the behaviors of the system are divided into two classes: the high-level behaviors executed by high-level users and the low-level behaviors executed by low-level users. Moreover, the system can be observed by users at different levels. In particular high-level behaviors should not be visible to low-level users. The system respects the non-interference property if the low-level behaviors are not affected by the high-level ones. In other words, if a system P acts in an environment where low-level and high-level users are present and are doing all that they can do, P is secure, i.e. non-interfering, if the observations that P offers to the low-level users when high-level behaviors are present and hidden are equal, in some suitable sense, to the ones that P offers when no high-level behavior is present.

The initial idea of non-interference was presented for a deterministic system represented by an automaton [GM82]. Since this first seminal paper, the idea has been fruitful in the framework of security, especially in the 90's.

It has been applied, for instance, to non-deterministic systems described by a

CCS-like ([Mil80]) process algebra, the Secure Process Algebra (SPA) of [FG96b, FG97, FG96a]. Different notions of non-interference are defined on this model and a classification defining an hierarchy between them based on the set of systems that each notion set as secure is given in [FG96b]. Other analyses based on process algebra can be found in [RS01, RWW96].

Other fields of application are analysis of security protocols and definition of security properties typical of this setting[FGG97], probabilistic systems [SS00] and information flows in a timed process algebra with a discrete time domain [FGM00].

Different notions of non-interference proposed in the literature have been compared and summarized in [FG96b, McL94]. An overview of some important questions concerning non-interference definitions, in a CSP setting, can be found in [RS01].

Usually, in a process algebra setting, the definition of non-interference proceeds as follows. The actions/events of a system are divided into high level and low level ones. Then, a restriction operator is defined on processes that forbid them to execute high-level actions. Moreover an hiding operator is defined that allows the execution of high-level actions but that make them not visible on the traces of the system which are the observable semantic objects. In this way, a process is said to be non-interfering if and only if the process resulting from the application of the restriction operator is equivalent, with respect to some suitable equivalence on traces or on the structure of the processes (i.e. bisimulation), to the process resulting from the application of the hiding operator.

This is equivalent to say that the view of the system of a low-level user, i.e. a user that observe only low-level actions, does not allow any deduction on the activity and even the presence of high-level users. This implies that covert channels do not exist in the system. Covert channels are communication flows from high-level users to low-level users through which secret information can be leaked using non-conventional ways, e.g. side effects of some legal operations.

The non-interference property of a system can be interpreted also in the following way. We can think of high-level activity as a dangerous activity for the system, e.g. the malicious actions of an intruder that wants to crash the system or some unusual legal operations that potentially make the system fail satisfying a requirement.

In this view, if the system is not interfering then we know that it is secure in the sense that an intruder cannot crash the system with its attacks or that the requirements are satisfied even in critical situations.

We mostly use the latter interpretation in the examples of analysis that we present in this chapter. However, obviously the timed non-interference that we define can also be applied using the other interpretation.

5.2 A Timed Notion of Non-Interference

In this section a notion of timed non-interference for real-time systems specified by timed automata is introduced. The notion is presented using an automata-based approach and then it is characterized also by operations and equivalence between timed languages. The definition is applied to an example of a time-critical system modeling a simplified control of an airplane.

To define our notion of non-interference, we partition the alphabet of the timed automaton representing a real-time system in two classes: high-level actions and low-level actions. Our notion of timed non-interference depends on a natural number n representing a minimum delay between high-level actions such that the low-level behaviors are not affected by the high-level ones. This means that this notion is suitable to detect interference due to high frequency of high-level actions. An intuitive example of this interference is the following. Consider a timed automaton T that models a speed-dependent real-time system like an airplane control system. It has to control a lot of basic events and has to respond with basic actions in order to maintain, say, the flight stability. These actions/events may be considered low-level actions and are always activated. When the pilot decides, for example, to turn right, he uses the cloche and this may correspond to an occurrence of a high-level action. Thus, the system receives high-level events (in this case cloche movements signals) separated by certain delays; *it must respond to them and must continue to catch and manage basic events*. When this happens we say that high-level actions delays magnitude does not affect the basic behavior of the system. Intuitively if the cloche movements signals are sent to the automaton with a too much high frequency, it could reach a state in which it is no longer able to manage basic events.

The new notion presented here is based on high-level actions delays magnitude and on trace equivalence of timed automata. Given a natural number n, we say that *high-level actions do not interfere with the system, considering a minimum delay n, if the system behavior in absence of high-level actions is equivalent to the system behavior, observed on low-level actions, when high-level actions can occur, but the delay between any two of them is greater than or equal to n.* Thus, if the system does not fire high-level events/actions separated by less than n time units and the property holds, there is no way for low-level users to detect any high-level activity or, in the other interpretation, high-level activity cannot interfere with the low-level one that is considered the only "desired" one. The main improvement with respect to the untimed notion of non-interference is that time is observable and the property can express security requirements of real-time systems, for instance the ones that follow:

- the time delay between high-level actions *cannot* be used to construct timed covert channels.

- the frequency of high-level actions does not affect the low level behaviors that are considered the correct ones.

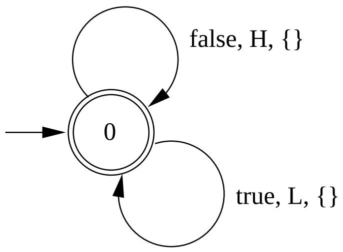

Figure 5.1: Inhib_H

5.2.1 n-non-interference

Let T be a timed automaton over an alphabet Σ. We remark that all the definitions and results that we give in this chapter can be applied to timed automata and also to timed automata with ϵ transitions. However we give them for timed automata. We suppose that Σ is partitioned into two disjoint sets of actions H and L: H is the set of high-level actions, while L is the set of low-level ones.

To observe the behavior of an automaton T in absence of high-level actions, we can compose T in parallel with an automaton, from now on called Inhib_H, that does not allow the execution of high-level actions.

The automaton Inhib_H is shown in Figure 5.1. In the parallel composition $T \parallel \mathrm{Inhib}_H$ all actions are synchronization actions. Thus, the component T cannot have a transition labeled by $\sigma \in H$ because his partner in synchronization, Inhib_H, never performs high-level actions (its constraints on high level actions are *false*). Thus only low-level actions are executed.

We have that $\mathcal{L}(T \parallel \mathrm{Inhib}_H)$ contains all basic behaviors of T, i.e. all timed words obtained by runs in which high-level actions do not occur.

Consider the automaton Interf^n_H in Figure 5.2. This automaton allows the execution of high-level actions only when they are separated by at least n time units. To see this, consider its behavior from the initial state i_0. There, it can perform low-level actions without restrictions. If never high-level actions occur in the run, then it stays forever in i_0 cycling on low-level actions. If a high-level action occurs the automaton changes its state to i_1 and reset a clock called x_{interf}. Note that this clock is reset by all high-level actions. This means that, in state i_1, x_{interf} always records the time elapsed from the previous high-level action occurred. In state i_1 low-level actions are allowed again with no restrictions, but any high-level action can be executed only if at least n time units have elapsed from the previous one. This is expressed by the constraint $x_{\mathrm{interf}} \geq n$.

Given an automaton T, the parallel composition $T \parallel \mathrm{Interf}^n_H$ can be used to observe the set of all behaviors of T such that high-level actions occur at times separated by an interval whose length is greater than, or equal to n. Of course, we are assuming that T does not reset the clock x_{interf}.

Before giving the definition of n-non-interference, we need the following oper-

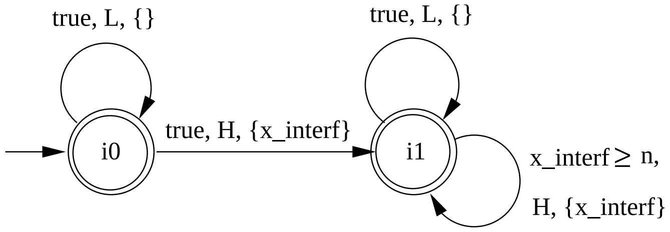

Figure 5.2: The structure of Interf_H^n

ation, which, if applied to an automaton T, returns an timed automaton with ϵ transitions having the same behaviors of T, but where high-level actions are non-observable.

Definition 5.1 (hiding of high-level actions) *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$. We denote by $T[H]$ the automaton obtained by T by replacing each edge $(q, \psi, \gamma, \sigma, q')$ of T with $\sigma \in H$, with the edge $(q, \psi, \gamma, \epsilon, q')$.*

Now we can define formally the intuitive notion of timed non-interference outlined above. Recall the definition of trace equivalence between timed automata that we have done in Definition 1.16.

Definition 5.2 (n-non-interference) *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$, and let $n \in \mathbb{N}$. High-level actions do not interfere in T with a minimum delay n (equivalently we say that T is n-non-interfering), if and only if*

$$(T \parallel \mathrm{Interf}_H^n)[H] \;\approx\; T \parallel \mathrm{Inhib}_H$$

5.2.2 Characterization with Timed Languages

In this section we characterize the notion of timed non-interference, defined in the previous section, using timed languages. We define three operations on timed languages and then we relate them to the corresponding operations on automata. This provides a more formal justification of the definition of n-non-interference of the previous section. As a matter of fact, it was given following some remarks on the behaviors of the automata Inhib_H and Interf_H^n when composed with the automaton T representing the system. The proofs of the following propositions are rather simple. We only give the proof of Proposition 5.2. The other ones are similar.

First, consider the restriction of a timed language to low-level actions.

Definition 5.3 (restriction to low-level actions) *Let I be a set of timed words over an alphabet $\Sigma = (H, L)$ (i.e. a timed language on Σ). $I|_L$ is the subset of I such that all elements are timed words on actions in L only:*

$$I|_L = \{(\overline{\sigma}, \overline{t}) \in I \mid \forall i \in \mathbb{N}.\ \sigma_i \in L\}$$

Proposition 5.1 *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$. Then $\mathcal{L}(T)|_L = \mathcal{L}(T \parallel \mathrm{Inhib}_H)$.*

The restriction to high level actions separated by at least n time units is defined as follows.

Definition 5.4 (n-delay restriction) *Let I be a set of timed words over an alphabet $\Sigma = (H, L)$. Let n be a natural number. I^n_H is the subset of I containing all timed words in I such that the delay between any two actions in H is at least n:*

$$I^n_H = \{(\overline{\sigma}, \overline{t}) \in I \mid \forall i, j \in \mathbb{N}.\ (i \neq j \wedge \sigma_i, \sigma_j \in H) \Rightarrow |t_i - t_j| \geq n\}$$

Proposition 5.2 *Let $T = (Q, \Sigma = (H, L), \mathcal{E}, B, R, X)$ be a timed automaton. Then $\mathcal{L}(T)^n_H = \mathcal{L}(T \parallel \mathrm{Interf}^n_H)$.*

Proof. Let w be a timed word in $\mathcal{L}(T)^n_H$. This means that $w \in \mathcal{L}(T)$ and it satisfies the condition expressed in Definition 5.4. Since w is accepted by T, there exists a run $r = (q_0, \nu_0) \xrightarrow{l_0} (q_1, \nu_1) \xrightarrow{l_1} \dots$ of T whose action sequence equals w (actually, there exist infinitely many runs having this property; we take just one of them). Let $\overline{q} \in (\mathit{inf}(r) \cap R)$ be a repeated state of T that is entered infinitely many times along r according to the Büchi acceptance condition.

We want to construct, from r, a run of $T \parallel \mathrm{Interf}^n_H$ whose action sequence equals w. There are two cases. The first one is when a high level action is never executed along r. In this case the run is just $r' = ((q_0, i_0, 0), \nu_0 \cup \{x_{\mathrm{interf}} = 0\}) \xrightarrow{l_0} ((q_1, i_0, m_1), \nu_1^*) \xrightarrow{l_1} \dots ((q_i, i_0, m_i), \nu_i^*) \xrightarrow{l_i} \dots$ where ν_i^* are the clock valuations ν_i of r extended with the valuation for the clock x_{interf} of Interf^n_H. Along r (and also r') this clock is never reset and Interf^n_H stays forever in state i_0 which is a repeated state. Thus, along the run so constructed, the state $(\overline{q}, i_0, 2)$ is taken infinitely many times satisfying the acceptance condition of the parallel composition (see Definition 1.12); moreover r' has the same label sequence of r and thus the corresponding accepted timed word is w.

The second case is when at least one high-level action h is executed along r. Suppose this happens at step k. Consider the prefix of a run $r' = ((q_0, i_0, 0), \nu_0 \cup \{x_{\mathrm{interf}} = 0\}) \xrightarrow{l_0} ((q_1, i_0, m_1), \nu_1^*) \xrightarrow{l_1} \cdots ((q_k, i_0, m_k), \nu_k^*) \xrightarrow{l_k = h} ((q_{k+1}, i_1, m_{k+1}), \nu_{k+1} \cup \{x_{\mathrm{interf}} = 0\})$ where ν_i^* are again the extensions of ν_i of r to handle the clock x_{interf}. If all Σ labels following l_k in r are low-level actions, we can extend r' as in the first case and we have that the state $(\overline{q}, i_1, 2)$ occurs infinitely many times in r', and thus r' is a run of $T \parallel \mathrm{Interf}^n_H$. If, on the other hand, other high-level actions occur in r, i.e. $r = \cdots (q_{k+1}, \nu_{k+1},) \xrightarrow{l_{k+1}} \cdots (q_j, \nu_j, m_j) \xrightarrow{l_j} (q_{j+1}, \nu_{j+1}) \cdots$, with $l_j \in H$ and for each $l_i, k < i < j$, $l_i \in L \cup \mathbb{R}^{>0}$, we can extend r' as follows: $r' = \cdots ((q_{k+1}, i_1, m_{k+1}), \nu_{k+1}^*) \xrightarrow{l_{k+1}} \cdots ((q_j, i_1, m_j), \nu_j^*,) \xrightarrow{l_j} ((q_{j+1}, i_1, m_{j+1}), \nu_{j+1}^*) \cdots$

Since r satisfies the condition of Definition 5.4, we have that at least n time units have elapsed between the occurrence of l_k and that of l_j. Now, since $\nu_{k+1}^* =$

$\nu_{k+1} \cup \{x_{\text{interf}} = 0\}$ and x_{interf} is not reset by low-level actions, then the value held by x_{interf} in ν_j^* is greater than, or equal to, n, and this means that r' is a prefix of a run of $T \parallel \text{Interf}_H^n$. Since $\nu_{j+1}^* = \nu_{j+1} \cup \{x_{\text{interf}} = 0\}$, the same argumentation can be used to show that the next high-level action of r (if any) can be executed in r', and so on. Thus, $w \in \mathcal{L}(T \parallel \text{Interf}_H^n)$. The converse can be easily proved by a similar argument. ■

The hiding of high-level actions is expressed as follows.

Definition 5.5 (hiding of high-level actions) *Let I be a set of timed words over an alphabet $\Sigma = (H, L)$. $I[H]$ contains the timed words $\overline{(\sigma, t)}$ of I in which the pairs (σ_i, t_i) with $\sigma_i \in H$ are discarded:*

$$I[H] = \left\{ \omega' \;\middle|\; \begin{array}{l} \omega = (\overline{\sigma}, \overline{t}) \in I \textit{ and } \omega' \textit{ is the projection of } \omega \\ \textit{on the pairs } \{(\sigma, t) \mid \sigma \in L\} \end{array} \right\}$$

Proposition 5.3 *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$. Then $\mathcal{L}(T)[H] = \mathcal{L}(T[H])$.*

The following proposition states that the above characterization correctly expresses n-non-interference.

Theorem 5.4 *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$, and $n \in \mathbb{N}$.*

$$T \textit{ is } n\textit{-non-interfering} \quad \textit{iff} \quad \mathcal{L}(T)_H^n[H] = \mathcal{L}(T)|_L$$

Proof. Follows from Definition 5.2 and the three propositions above. ■

The following proposition shows that the frequency of high-level actions, that becomes higher as n decreases, is a crucial parameter on determining the n-non-interference property.

Proposition 5.5 *Let T be a timed automaton and $n \in \mathbb{N}$. If T is n-non-interfering then T is also $(n + 1)$-non-interfering.*

Proof. It is simple to see that the condition of n-non-interference depends on the frequency of the occurrence of high-level actions. More precisely, it can be missed only if such a frequency is too high. Thus, if the condition holds for a certain n it will hold for every $n' \geq n$ because the n'-non-interference requires the same equivalence allowing at most a lower frequency of high-level actions than the n-non-interference, which is verified by hypothesis. ■

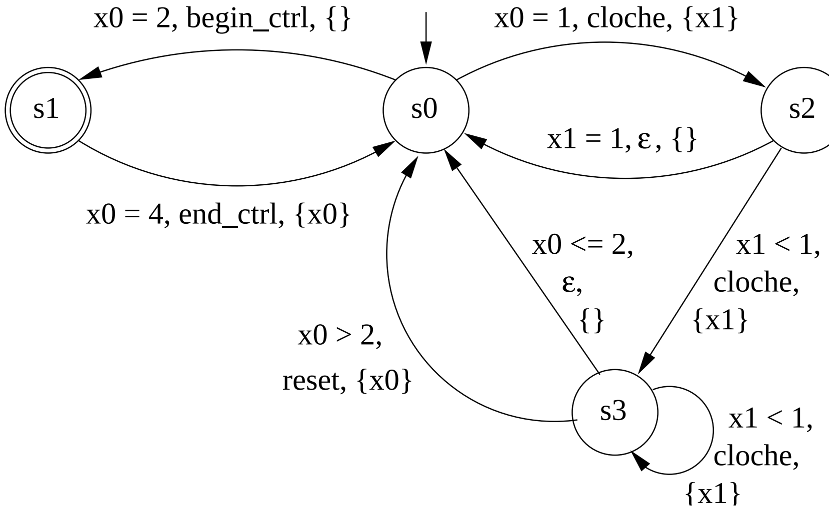

Figure 5.3: Automaton T: a simplified airplane control

5.2.3 An example

In this section we consider a simple control T of an airplane (Figure 5.3) and study its non-interference properties. The system periodically executes, at predefined instants, a set of operations to control flight stability. The control operations begin with action *begin_ctrl* and end with action *end_ctrl*. State *s1* is an abstraction for the control operations, which require 2 time units to be completed. One time unit before entering each control cycle the system can catch an input action from the pilot; in this simple case we consider only a single input action *cloche* modeling cloche movements. When *cloche* occurs, the system handles it and then continues to manage control actions. Let *begin_ctrl* and *end_ctrl* be low-level actions and *cloche* and *reset* be the high-level actions.

At first consider the behavior of T when the high-level action *cloche* is disabled (i.e. the moves of $T \parallel \mathrm{Inhib}_H$); this consists in a simple cycle between states *s0* and *s1* where transitions are separated by exactly 2 time units.

Thus $\mathcal{L}(T \parallel \mathrm{Inhib}_H)$ contains a single timed word

$$(begin\_ctrl, 2)(end\_ctrl, 4)\cdots(begin\_ctrl, 2+4i)(end\_ctrl, 4(i+1))\cdots \tag{5.1}$$

Consider now the general behavior of the system, i.e. all actions are enabled with no restrictions. When *cloche* occurs, the system moves to state *s2*. When the system is in *s2* it is handling the *cloche* action. This operation normally requires one time unit, after which the system returns to the initial state. However, while being in *s2*, the system can catch other *cloche* actions and in this case it moves to state *s3*. If the handling of them requires too much time ($x0$ becomes greater than 2), it is necessary to postpone the beginning of the successive control cycle (i.e. action *reset* is executed).

Thus, the *cloche* actions can interfere with the basic system behavior if they are too close to each other. To see this formally with our approach to non-interference consider a natural number n and the automaton $T \parallel \mathrm{Interf}^n_H$. If $n = 0$ high-level actions can occur without restrictions on their relative delays and consequently the *reset* action could be executed. Thus, the system does not satisfy our non-interference definition. If, instead, $n \geq 1$, then only one *cloche* action can occur between two successive control cycles and it is managed without affecting the basic behavior. Thus, for each $n \geq 1$, $(T \parallel \mathrm{Interf}^n_H)[H]$ generates the single timed word (5.1) which in turn is the unique timed word generated by $T \parallel \mathrm{Inhib}_H$. From Definition 5.2 we conclude that high-level actions do not interfere in T with a minimum delay $n \geq 1$ (T is n-non-interfering for $n \geq 1$). On the other hand, if $n = 0$ there is interference.

5.3 Toward Decidability

In order that the theory presented so far can become useful in practice, we must check the trace equivalence of timed automata. This implies checking inclusion in both directions of the respective recognized languages. We have seen in Section 2.2.2 that for timed automata the language inclusion problem is undecidable. However, the language inclusion problem can be solved if the system can be modeled using deterministic Muller automata (Section 1.4.3) or event-clock automata [AFH97]. To make the method effective also in the general case, two directions can be followed. On one side weaker notions of non-interference which are checkable on general timed automata can be defined. On the other side, sufficient conditions on the structure of the automaton T representing the system, which allow us to decide the equivalence check required by our timed non-interference definition can be searched for.

We find that the former direction is more promising and, since the objective of decidability is more appeal if we get also effectiveness of the verification tasks, our simplification of the timed non-interference naturally follows the direction toward timed safety automata (Section 2.3.1).

We know that time safety automata has state-base semantics while timed automata, as usual in the automata-theoretic approach, has a trace-based semantics, i.e. timed words. A natural direction to simplify the n-non-interference is the definition of a state-based notion. We know, from Section 2.2.2, that the reachability of states is decidable. Moreover, all tools for the verifications of timed (safety) automata implement algorithms to perform reachability analysis, which is the most studied from the point of view of efficiency. This is because a lot of real-time requirements can be reduced to a reachability test (see Section 2.4.2).

5.4 A Decidable Timed non-Interference

We consider an equivalence between timed automata which is decidable and it is used to define a decidable notion of timed non-interference derived from n-non-interference. The equivalence focuses the reachable states of the system. The decidability of the new notion relies on the decidability of the reachability test for timed (safety) automata. The definition is as follows.

Definition 5.6 **(n-state-non-interference)**
Let $T = (Q, \Sigma, \mathcal{E}, B, R, \mathcal{X})$ be a timed automaton with an alphabet $\Sigma = H \cup L$, where $H \cap L = \emptyset$, and let n be a natural number. T is n-state-non-interfering *iff the set of reachable states of $T \parallel \mathrm{Inhib}_H$, projected on the states Q of T, is equal to the set of reachable states of $(T \parallel \mathrm{Interf}^n_H)[H]$, projected on the states Q of T.*

Let $R(T)$ be the set of reachable states of $T \parallel \mathrm{Inhib}_H$, projected on the states Q of T. Since this automaton can not perform high level actions, $R(T)$ contains those states of T that can be accessed without the aid of any high level activity. The notion of n-state-non-interference then requires that $R(T)$ does not change when a controlled high-level activity is allowed. This control imposes that any two subsequent high level actions are separated by at least n time units.

Note that we have defined the n-state-non-interference using timed automata. It is straightforward to adapt this definition to the model of timed safety automata.

The following proposition shows that the two notions of non-interference take different aspects of systems into account.

Proposition 5.6 *Let T be a timed automaton.*

1. T is n-non-interfering $\not\Rightarrow$ T is n-state-non-interfering.

2. T is n-state-non-interfering $\not\Rightarrow$ T is n-non-interfering.

Proof. We will present two counter examples. Figure 5.4 shows an automaton which is n-non-interfering for all $n \in \mathbb{N}$. This is because the high level transition labeled with h from state 0 to state 2 is followed, possibly without delay, by a transition labeled with the low level action l from state 2 to state 1. Thus, when the automaton $T \parallel \mathrm{Inhib}_H$ is considered, the action h can not be performed and the accepted timed words are of the form $(l, t_0)(l, t_1) \cdots$ in which it is only required that $t_i \leq t_{i+1}$. When the automaton $(T \parallel \mathrm{Interf}^n_H)[H]$ is considered, the action h can be performed, but is not observable. Thus the accepted timed words are of the same form of those of $T \parallel \mathrm{Inhib}_H$ because there are not clock constraints. On the other hand, the system T of Figure 5.4 is not n-state-non-interfering for any $n \in \mathbb{N}$. This is because the state 2 is reachable in $(T \parallel \mathrm{Interf}^n_H)[H]$ for all $n \in \mathbb{N}$ and it is not reachable in $T \parallel \mathrm{Inhib}_H$.

Figure 5.5 shows the counterexample for the converse. In this case it is clear that the automaton is n-state-non-interfering. On the other hand, all the accepted timed

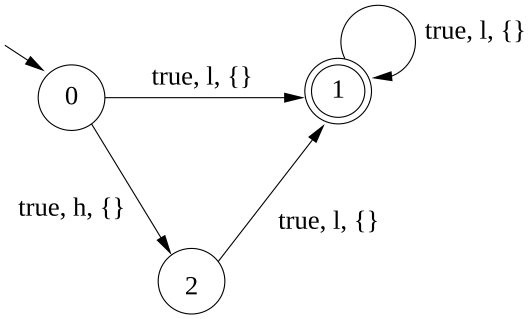

Figure 5.4: A system that is n-non-interfering, but not n-state-non-interfering

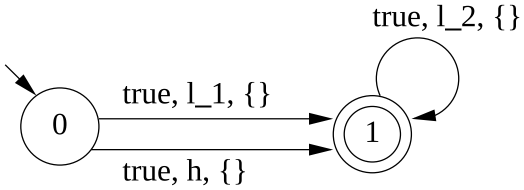

Figure 5.5: A system that is n-state-non-interfering, but not n-non-interfering

words of $T \parallel \text{Inhib}_H$ begin with $(l_0, t_0)\cdots$, but the automaton $(T \parallel \text{Interf}^n_H)[H]$ accepts a timed word beginning with $(l_1, t_0)\cdots$. ■

5.5 The Fischer's Protocol for Mutual Exclusion

In this section we show the utility of n-state-non-interference analysis to state the robustness, with respect to external attacks, of a classical timing-based mutual exclusion protocol. The protocol was suggested by Michael Fischer and reported in [Lam87]. The significance of the protocol is due to its speed, that makes it suitable for multiprocessor computers or time-critical embedded systems.

Suppose that two processes, P_1 and P_2, are running in parallel, competing for a critical section, and assume that atomic reads and writes are permitted to a shared variable x. Assume also that every access to the shared memory containing x takes *acc* units of time. Each process executes the following algorithm, where the code both of the critical section and the one outside the protocol is assumed not to modify x.

```
repeat
       await x=0;
       x:=i;
       delay b
until  x=i;
```

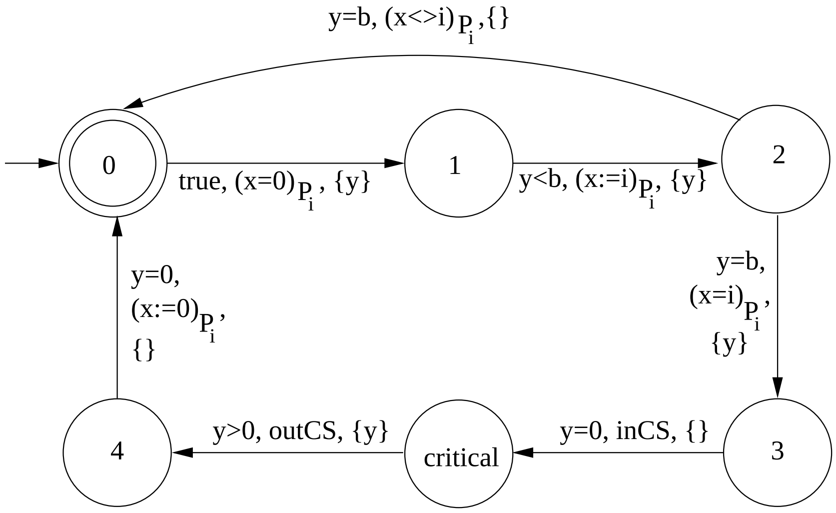

Figure 5.6: Automaton P_i

```
Critical Section;
x:=0
```

Each process P_i is allowed to be in its critical section iff $x = i$. The statement `await x=0` waits until the value of x becomes 0. The statement `delay b` delays a process for b time units, as measured by the process clock. Here we assume that the local clocks of the two processes proceed at the same rate. Each statement takes an amount of time to be executed, in particular we assume that the assignment statement takes at most a time units. Recall that an access to the shared memory containing x is atomic, and the processes P_1 and P_2 can compete for accessing it, thus the time of each assignment, a, may depend on the resolution of conflicts.

The Fisher's protocol ensures mutual exclusion iff $a < b$.

Each process can be represented by a timed automaton like the one in Figure 5.6, a slightly different representation of the one given in [AJ98, ACH$^+$95].

State 0 corresponds to the local computation of the process. x is not a clock variable: it represents the shared variable of the protocol. y is a clock variable that is used to count time as specified in the protocol specification. The process can start the protocol for accessing the critical section only if the value of x is equal to 0. This is represented by the $(x = 0)_{P_i}$ action: a synchronization action with the `Serializer` (see Figure 5.7). At this point (state 1) it can assign x (in a time shorter than b[1]) and it waits b time units for testing it, and, depending on its value, for entering the critical region (state *critical*). On exiting the critical section, the process sets the value of x to 0.

It is important to note that P_1 and P_2 must not synchronize on actions, but only

[1] Actually the assignment has to occur within a time units and a has to be less than b. Here, for simplicity, we use b in both cases.

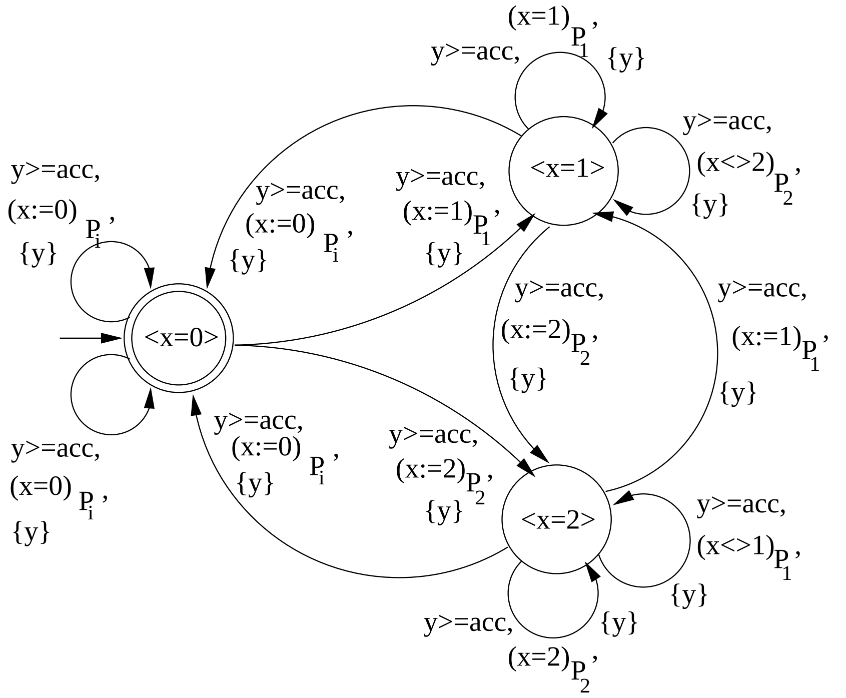

Figure 5.7: The `Serializer`

on the value of variable x. Thus the same action executed by a process is considered different if executed by another process; for instance $(\texttt{x:=0})_{P_1}$, executed by P_1, is different from $(\texttt{x:=0})_{P_2}$, executed by P_2.

The automaton of Figure 5.6 does not consider the time, *acc*, for accessing the shared memory. Thus, when P_1 and P_2 are combined in parallel they can both start an access to x, and the time interval between these accesses could be shorter then *acc*. This is in contrast with the assumption that accesses to x are atomic.

To force the accesses to be atomic and to control the value of the variable x, we add, to the system composed by the two processes, the `Serializer` of Figure 5.7 which synchronizes with P_1 and P_2 on every action which performs an access to x. The clock y is used to ensure that every access on the variable x (test and/or assignment) is performed after at least *acc* time units since the last one. This assures atomicity. Note that the states of the automaton of Figure 5.7 are associated with the three possible values of variable x. The actions $(x <> i)_{P_i}$ and $(x = i)_{P_i}$ correspond to the test of $x \neq i$ or $x = i$ respectively. They can be taken by the process P_i only if the test is true, according to the information on the value of x held in the current state of the `Serializer`.

Let us analyze the behavior of the two processes, P_1 and P_2. The automaton $(P_1 \parallel P_2 \parallel \texttt{Serializer})$, obtained by the synchronized product of the components, can reach any possible state $\langle s_1, s_2, s\rangle$[2] (where s_1 and s_2 are states of P_1 and P_2, and

[2]For readability here and in the following we ignore the counters in the states introduced by

s is a state of the `Serializer`) but the states $\langle critical, critical, s\rangle$, because of the correctness of the mutual exclusion protocol.

Let us assume now that there is a upper bound, ucs, on the time needed to execute the critical section after the variable x is checked. This assumption is reasonable when the operations in the critical section are fast and simple, for example the ones for updating a counter. If $ucs \leq a+b$, the previous protocol can be modified in order to decrease the delay in accessing the critical section when a conflict is present. The idea is that, because $ucs \leq a + b$, after executing successfully the protocol and before entering the critical section, a process can signal to the other that the protocol can be executed again. This is safe because the time for executing the critical section is less than or equal to the time taken by the protocol itself. Actually ucs includes the time for assigning the value 0 to x.

The new protocol is now the following.

```
repeat
       await x=0;
       x:=i;
       delay b
until  x=i;
x:=0
Critical Section;
```

Let us represent process P_i' executing the new protocol by the timed automaton of Figure 5.8.

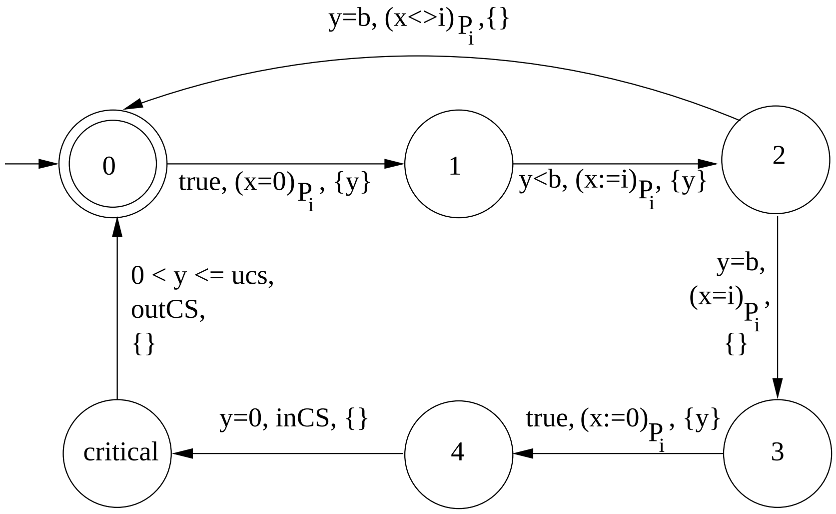

Figure 5.8: Automaton P_i'

---

the parallel composition operator of timed automata and retain only the states of the components.

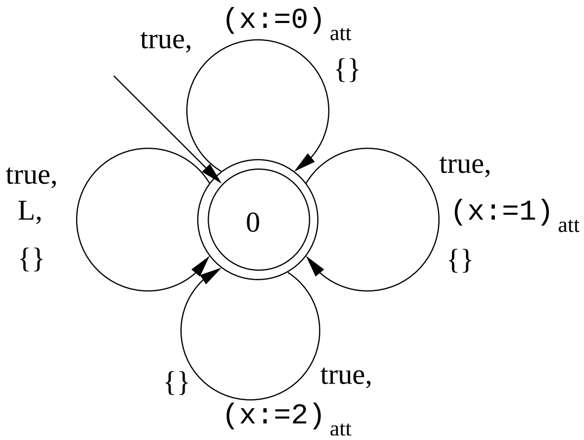

Figure 5.9: The `Intruder`

5.6 An n-state-non-Interference Analysis

Suppose the existence of an `Intruder` which is able to read and write the shared variable x. Such an `Intruder` can be either a malicious host connected to the network, or simply another component of the system accessing the shared memory for other purposes. The possible behaviors of the `Intruder` are described by the timed automaton in Figure 5.9[3]. Note that to implement atomic accesses to x the `Serializer` must be extended for taking account also of the actions of the `Intruder`. The modified `Serializer` is denoted by `Serializer`$'$.

Consider now the automaton $T = (P_1' \parallel P_2' \parallel \texttt{Serializer}' \parallel \texttt{Intruder})$

In order to apply Definition 5.6 we consider the writing actions of the `Intruder` as the high-level actions, thus $H = \{(\texttt{x} := 0)_{\text{att}}, (\texttt{x} := 1)_{\text{att}}, (\texttt{x} := 2)_{\text{att}}\}$. We use L to denote all other actions of the system.

Note that we have added an edge labeled with L without constraints and reset to the state of the `Intruder` (Figure 5.9). This is a technical trick needed because the parallel composition operator requires that every run is such that every component has its Büchi acceptance condition satisfied. Indeed, when composing the whole system with Inhib_H for the non-interference analysis, the parallel composition would accept the empty language if the edge is not added.

We want to show that T is n-state-non-interfering, for some n. This corresponds to show that the set of reachable states of the system $(P_1' \parallel P_2' \parallel \texttt{Serializer}' \parallel \texttt{Intruder} \parallel \text{Interf}^n_H)[H]$, projected on the states of $P_1' \parallel P_2' \parallel \texttt{Serializer}' \parallel \texttt{Intruder}$, is equal to the set of reachable states of the system in which all the actions of the `Intruder` are forbidden. We know, by the previous analysis of the system without the `Intruder`, that the states $\langle critical, critical, s\rangle$ (where s is any state of the `Serializer`) are not reachable. Thus, in this example, the condition of Definition 5.6 reduces to verify that the states $\langle critical, critical, s, 0\rangle$ of

[3]Actually only writing actions are critical, thus we restrict the system to them.

P_1'	`x=0`		`x:=1`					`x=1`			`inCS` `x:=0`		crash
P_2'		`x=0`			`x:=2`					`x=2`		`inCS` `x:=0`	crash
Intr							`x:=1`		`x:=2`				

Figure 5.10: An attack to the protocol

$(P_1' \parallel P_2' \parallel$ `Serializer'` $\parallel$ `Intruder` $\parallel \mathrm{Interf}_H^n)[H]$ (projected on the states of $(P_1' \parallel P_2' \parallel$ `Serializer'` $\parallel$ `Intruder`$)$) are still unreachable.

Remark that the original Fischer protocol was very weak from this point. If the `Intruder` set the variable `x` to 0 when a process is in its critical region, the other process is enabled to enter the critical region too.

First let us show that if the value of n is lower enough the system can be n-state-interfering, as showed in Figure 5.10, where $acc = 1$, $b = 6$, $ucs = 6$ and $n = 2$. Each depicted interval corresponds to acc time units no matter its graphical length.

Observing the attack to the protocol reported in Figure 5.10 we get some hints to obtain a general result. If $ucs \leq a + b$ and $a < b$ (these are the conditions needed by the protocol for its correctness in absence of attacks), we can conclude the following.

Proposition 5.7 *For all $n > b$ the system $P_1' \parallel P_2' \parallel$ `Serializer'` $\parallel$ `Intruder` is n-state-non-interfering.*

Proof. Note that, in order to break the protocol, the `Intruder`, when present, has to perform two successive assignments to x (i.e. two subsequent high level actions in our formulation) in a limited time interval. To see this, suppose that both P_1' and P_2' start a session of the protocol and they have reached their states 2. Suppose, at this point, that $x = 2$. This means that the first process that started the session was P_1'. The normal behavior at this point would be that P_1', seeing $x = 2$, returns to its idle state (state 1). But suppose that, here, the attacker sets $x := 1$ after P_2' entered its state 2 and reset its clock y ($y_{P_2'}$). Now, the session proceeds and P_1', when its clock y equals b, enter its state 3 because the attacker has set x to 1. For now on P_1' proceeds without any further control toward the critical section. Let us return to P_2'. It was waiting b time units before testing the value of x and deciding whether enter the critical section or not. Now the `Intruder` has to set $x := 2$ before P_2' test the value (when $y_{P_2'}$ equals b) because currently x equals 1. The `Intruder`, to break the protocol, should be allowed to do such an assignment. Surely it is not allowed if $n > b$ because the clock `x_interf` (the clock of Interf_H^n constraining the occurrence of high level actions) was reset after clock $y_{P_2'}$ was reset. Thus `x_interf` is less then

$y_{P_2'}$. To do the assignment (an high level action) `x_interf` must be greater than or equal to n and to break the protocol the assignment has to be done before $y_{P_2'}$ equals b. That is, it must hold `x_interf` $< y_{P_2'} < b$ and `x_interf` $\geq n$. This is not possible if $n > b$.

The previous argumentation does not assume anything about the value of b. Thus, if $n > b$, the set of reachable states of the system when the `Intruder` acts with the constraints expressed by Interf_H^n is equal to the set of reachable states of the system when the `Intruder` does not perform any action. ■

Concluding, by n-state-non-interference analysis, we are able to obtain an upper-bound on the frequency of attacks to the protocol under which it is not affected by the attack. Note that n-state-non-interference does not guarantee that the `Intruder` could not interfere in another manner, for instance causing a deadlock or forbidding a process to reach the critical section all of the times. These other attacks do not strongly depend on the frequency of the action of the `Intruder` and should be addressed using different methods.

5.7 Analysis with UPPAAL

In this section we model the system $P_1 \parallel P_2 \parallel$ `Serializer'` $\parallel$ `Intruder` using timed safety automata as they are specified in UPPAAL. Using this tool we automatically perform the verification of the previous section and we get evidence also of the Proposition 5.7. In Figures 5.11 and 5.12 are shown the processes P_1 and P_2. Note that we use the original formulation of the Fisher's protocol in this analysis.

Recall from Section 2.4.2 that integer variables can be defined in timed processes of UPPAAL and their values can be tested in guards of the transitions and can be assigned when a transition is performed. We use a global variable x that can be have values in the set $\{0, 1, 2\}$. This simplifies the `Serializer`' which is modeled as in Figure 5.13.

The `Intruder` is modeled in Figure 5.14. Note that it is not required any more in this setting that it perform also low-level actions.

In the following there are the constant and variable declarations for the system.

```
const acc 1;      /* Access time to x */
const b 6;        /* Parameter of the Fisher's Protocol */
const n 5;        /* Speed of the Intruder */
const ucs 6;      /* Maximal Time spent in Critical Section */
int[0,2] x:=0;    /* Share Variable */
chan ac, bc, cc;  /* Synchronization Channels */
```

The following symmetric properties are satisfied by this system:

$$\forall\Box(P_1.\texttt{critical} \rightarrow \neg P_2.\texttt{critical})$$

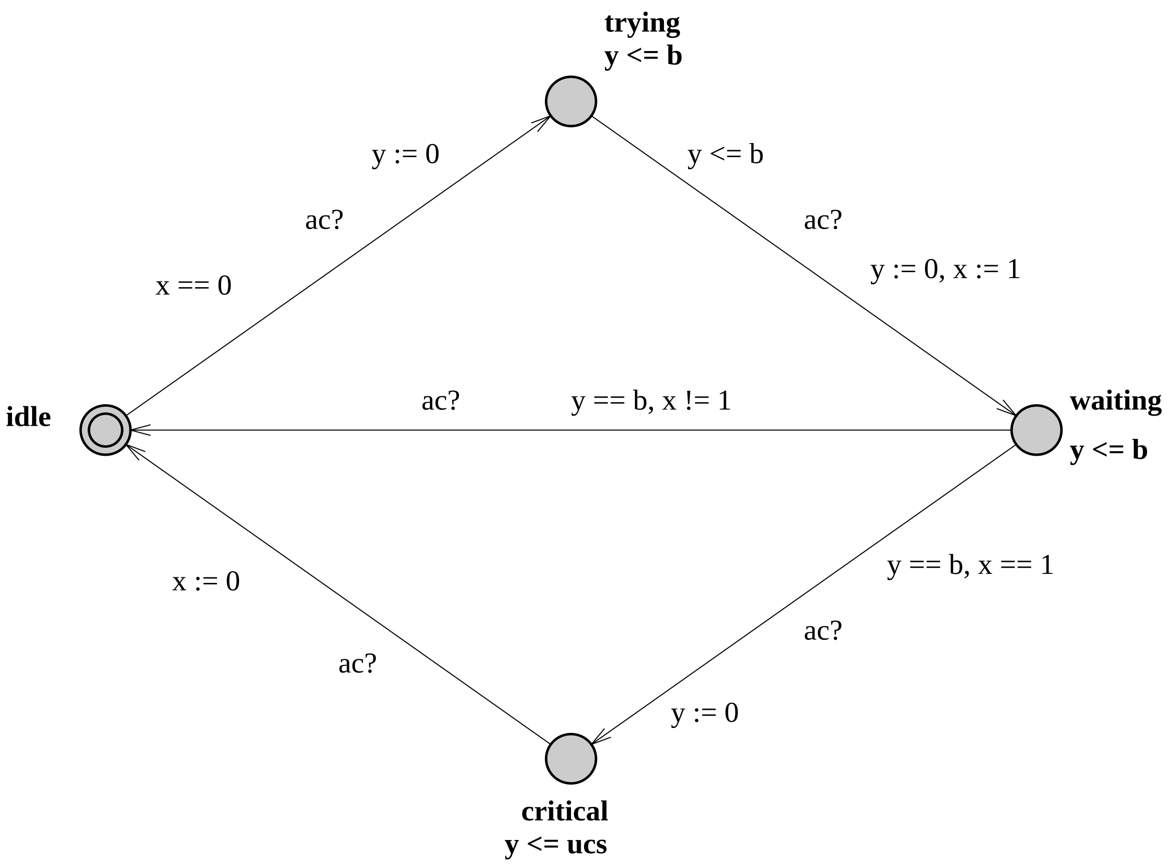

Figure 5.11: Process P_1 modeled in UPPAAL

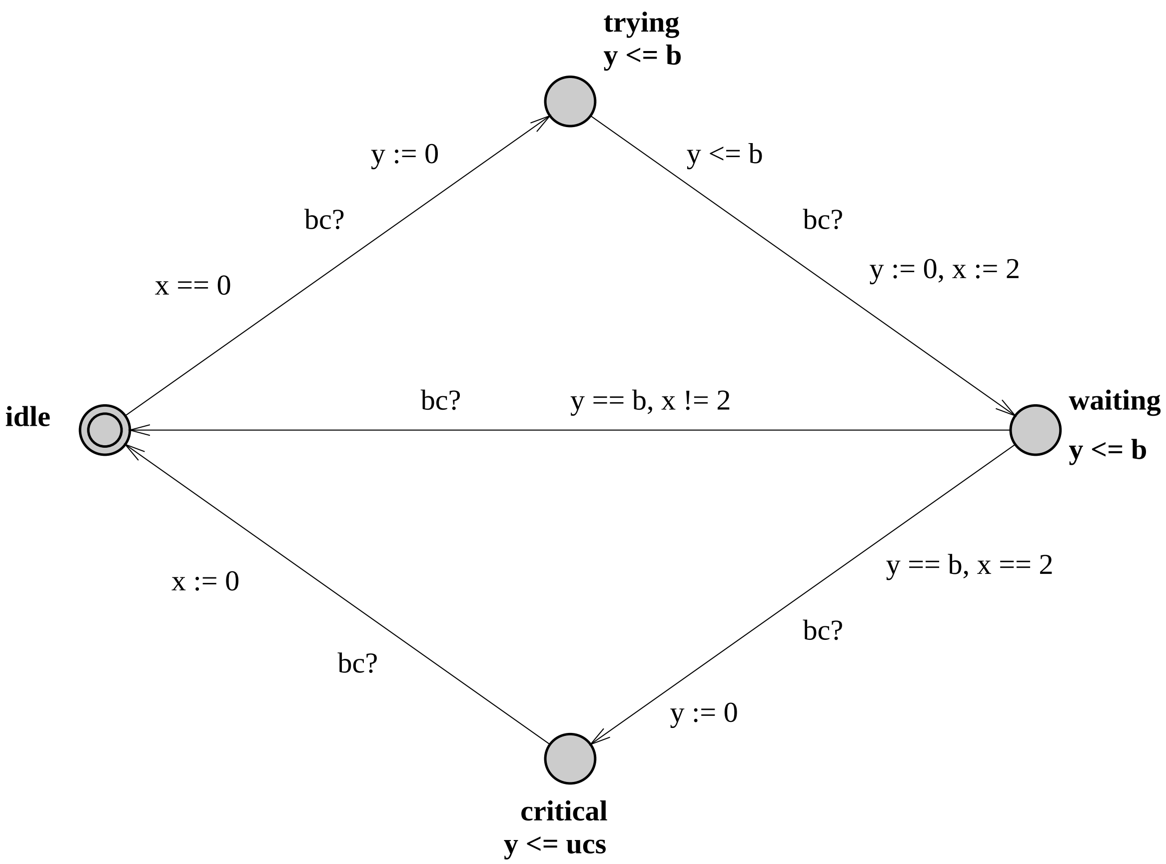

Figure 5.12: Process P_2 modeled in UPPAAL

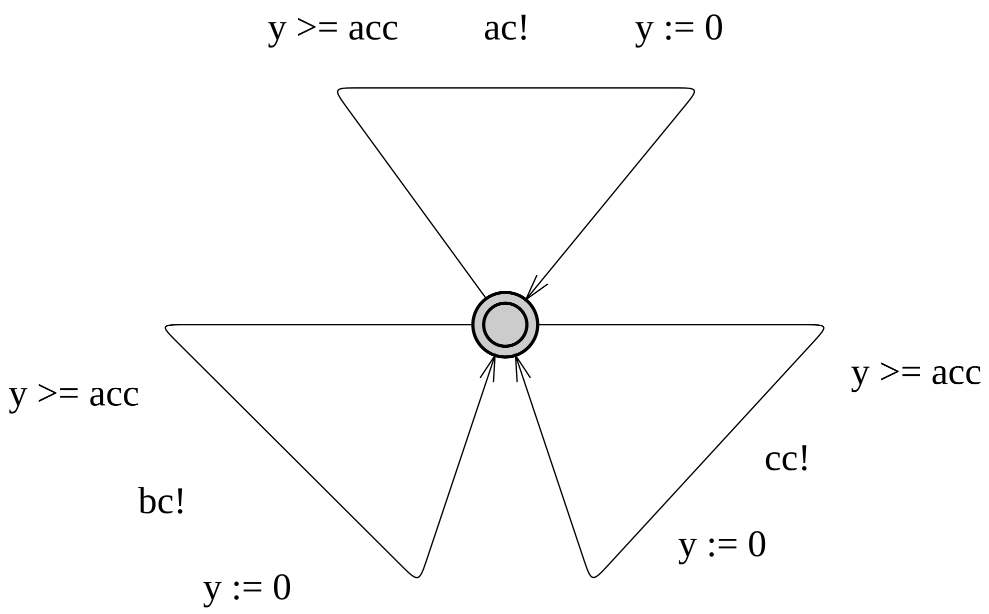

Figure 5.13: Process `Serializer` modeled in UPPAAL

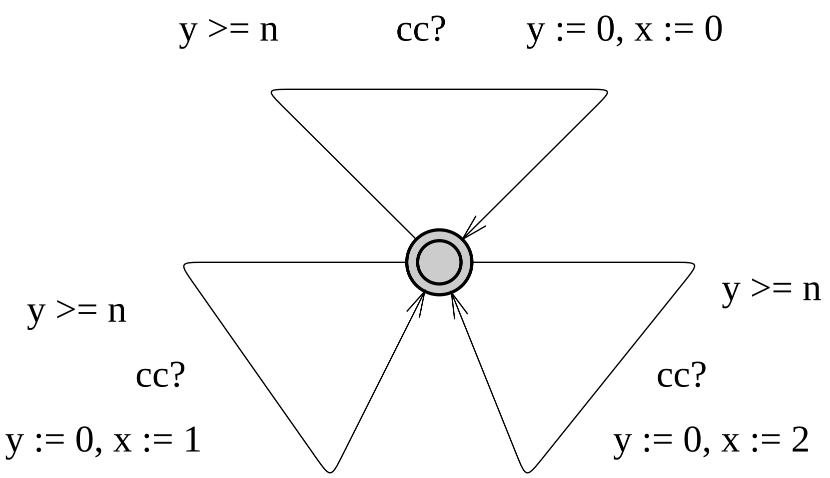

Figure 5.14: Process `Intruder` modeled in UPPAAL

$$\forall\Box(P_2.\texttt{critical} \rightarrow \neg P_1.\texttt{critical})$$

This means that the system is 5-state-non-interfering. If we increase the frequency of the high-level actions of the `Intruder` setting $n = 4$, the properties are not satisfied and the diagnostic trace given by UPPAAL is equal to the one specified in Figure 5.10 of the previous section, but some differences on the times. Of course the system is still n-state-interfering if n is decreased further ($n \leq 4$) and it is n-state-non-interfering for all $n > 4$.

Conclusions

We remark that Chapters 1 and 2 are an important contribution of this thesis, as well as introductory parts. We have given a detailed and modular introduction of the timed automata model trying to individuate a synthesis for the different variants of this formalism that have been introduced in the literature. The main theoretical results for the verifications using timed automata have been recalled trying to explain clearly the fundamental steps. The *implementable* timed automata, called timed safety automata, are defined staying as close as possible to the definition of timed transition tables. Moreover, the semantic models of timed safety automata are used to specify the semantics of the timed temporal logic TCTL.

The extensions introduced in Chapters 3 and 4 give a contribution in the area of specification using timed automata and Chapter 5 contributes in the area of verification.

Future work in these topics can be individuated mostly in the implementation area. We have outlined how to define the constructions of Chapters 3 and 4 also for timed safety automata and a tool realizing this task could be an interesting application. This is true especially for the construction of urgency where the zone automaton should be used instead of the region automaton.

Chapter 5 can be considered the starting point of an interesting research direction. We think that other notions of timed non-interference can be defined varying both the equivalence that is used (for instance timed-abstract bisimulations could be used instead of reachable-states equivalence) and the type of hiding and restriction operators.

Bibliography

[ABBL98] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. In *Proceedings of the 18th Conference on Foundations Software Technology and Theoretical Computer Science (FSTTCS'98)*, number 1530 in Lecture Notes in Computer Science, pages 245–256. Springer, Berlin, 1998.

[ABDS00] G. Anastasi, A. Bartoli, N. De Francesco, and A. Santone. Efficient verification of a multicast protocol for mobile computing. *The Computer Journal*, 44:21–30, 2000.

[ABG98] L. Aceto, A. Burgueño, and K. Guldstrand Larsen. Model checking via reachability testing for timed automata. In *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, number 1384 in Lecture Notes in Computer Science, pages 263–280. Springer, Berlin, 1998.

[ACD93] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[ACD94] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. *Theoretical Computer Science*, 126:183–236, 1994.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *In Proceedings of the 3rd International Conference on Concurrency Theory (CONCUR'92)*, number 630 in Lecture Notes in Computer Science, pages 340–354. Springer, Berlin, 1992.

[ACH94] R. Alur, C. Courcoubetis, and T. A. Henzinger. The observational power of clocks. In *Proceedings of the 5th International Conference on Concurrency Theory (CONCUR'94)*, number 836 in Lecture Notes in Computer Science, pages 162–177. Springer, Berlin, 1994.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, number 736 in Lecture Notes in Computer Science. Springer, Berlin, 1993.

[ACM97] E. Asarin, P. Caspi, and O. Maler. A kleene theorem for timed automata. In *Proceedings of the 12th IEEE Symposium on Logic in Computer Science (LICS'97)*, pages 160–171. IEEE Computer Society Press, 1997.

[ACM02] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of the ACM*, 49(2):172–206, 2002.

[AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proceedings of the 17th Colloquium on Automata, Languages and Programming (ICALP'90)*, number 443 in Lecture Notes in Computer Science, pages 322–335. Springer, Berlin, 1990.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ADS86] B. Alpern, A. Demers, and F. Shneider. Safety without stuttering. *Information Processing Letters*, 23(4):177–180, 1986.

[AFH96] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[AFH97] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *???*, 1997.

[AFH99] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[AH89] R. Alur and T. A. Henzinger. A really temporal logic. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS'89)*, pages 164–169. IEEE Computer Society Press, 1989.

[AH90] R. Alur and T. A. Henzinger. Real-time logics: complexity and expressiveness. In *Proceedings of the 5th Annual IEEE Symposium in Logic and Computer Science (LICS'90)*, pages 390–401. IEEE Computer Society Press, 1990.

[AH92a] R. Alur and T. A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *Proceedings of the 33th Annual IEEE Symposium on Foundations of Computer Science (FOCS'92)*, pages 177–186. IEEE Computer Society Press, 1992.

[AH92b] R. Alur and T. A. Henzinger. Logics and models of real-time: A survey. In *Real-time: Theory in Practice*, number 600 in Lecture Notes in Computer Science, pages 74–106. Springer, Berlin, 1992.

[AH94] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41:181–204, 1994.

[AJ98] P. A. Abdulla and B. Jonsson. Networks of timed processes. In *In Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, number 1384 in Lecture Notes in Computer Science, pages 298–312. Springer, Berlin, 1998.

[Alu99] R. Alur. Timed automata. In *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 8–22. Springer, Berlin, 1999.

[B̈60] J. R. Büchi. A decision method in restricted second order arithmetic. In *Proceedings of International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1960.

[B́95] B. Bérard. Untiming timed languages. *Information Processing Letters*, 55:129–135, 1995.

[BAL02] Mikael Buchholtz, Jacob Andersen, and Hans Henrik Lovengreen. Towards a process algebra for shared processors. In Flavio Corradini and Walter Vogler, editors, *Electronic Notes in Theoretical Computer Science*, volume 52. Elsevier, 2002.

[BDST01] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesei. A notion of non-interference for timed automata. In Ludwik Czaja, editor, *Proceedings of the Concurrency, Specification & Programming Workshop (CS&P'2001)*, pages 6–15. Warsaw University Press, October 2001.

[BDST02] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesei. A notion of non-interference for timed automata. *Fundamenta Informaticae*, 51(1-2):1–11, 2002.

[BDT00] Roberto Barbuti, Nicoletta De Francesco, and Luca Tesei. Timed automata with non-instantaneous actions. In *Proceedings of the Concurrency, Specification & Programming Workshop (CS&P'2000)*. Informatik-Berichte, Berlin, 2000.

[BDT01] Roberto Barbuti, Nicoletta De Francesco, and Luca Tesei. Timed automata with non-instantaneous actions. *Fundamenta Informaticae*, 47(3-4):189–200, 2001.

[BGP96] B. Bérard, P. Gastin, and A. Petit. On the power of non observable actions in timed automata. In *Proceedings of the 13th International Symposium on Theoretical Aspects of Computer Science (STACS'96)*, number 1046 in Lecture Notes in Computer Science, pages 257–268. Springer, Berlin, 1996.

[BL92] T. Bolognesi and F. Lucidi. Timed process algebras with urgent interactions and a unique powerful binary operator. In *Proceedings of Real-Time: Theory in Practice Workshop (REX Workshop 1991)*, number 600 in Lecture Notes in Computer Science, pages 124–148. Springer, Berlin, 1992.

[BL97] P. Brémond-Grégoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1-2):179–219, 1997.

[BLL+96] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL – a tool suite for automatic verification of real-time systems. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer, Berlin, 1996.

[BP99] P. Bouyer and A. Petit. Decomposition and composition of timed automata. In *Proceedings of the 26th Colloquium on Automata, Languages and Programming (ICALP'99)*, number 1644 in Lecture Notes in Computer Science, pages 210–219. Springer, Berlin, 1999.

[BPDG98] B. Bérard, A. Petit, V. Diekert, and P. Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36:145–182, 1998.

[BPT03] Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.

[BS97] S. Bornot and J. Sifakis. Relating time progress and deadlines in hybrid systems. In *Proceedings of International Workshop on Hybrid and Real-Time Systems (HART'97)*, number 1201 in Lecture Notes in Computer Science, pages 286–300. Springer, Berlin, 1997.

[BS00] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 163(1):172–202, 2000.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Proceedings of Compositionality Meeting (COMPOS'97)*, number 1536 in Lecture Notes in Computer Science, pages 103–129. Springer, Berlin, 1998.

[BT01] Roberto Barbuti and Luca Tesei. Timed automata with urgent transitions. In Flavio Corradini and Walter Vogler, editors, *Proceedings of the 2nd International Workshop on Models for Time-Critical systems (MTCS'01)*, number NS-01-5 in BRICS Notes, pages 3–21. Department of Computer Science, Aarhus University Press, 2001.

[BT02] Roberto Barbuti and Luca Tesei. A decidable notion of timed non-interference. In *Proceedings of the Concurrency, Specification & Programming Workshop (CS&P'2002)*, pages 20–38. Humboldt University, Informatik-Berichte, Berlin, 2002.

[BT03] Roberto Barbuti and Luca Tesei. A decidable notion of timed non-interference. *Fundamenta Informaticae*, 54(2-3):177–150, 2003.

[BT04] Roberto Barbuti and Luca Tesei. Timed automata with urgent transitions. *To appear on Acta Informatica*, 2004.

[CES86] E. Clarke, E. A. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CG00] C. Choffrut and M. Goldwurm. Timed automata with periodic clock constraints. *Journal of Automata, Languages and Combinatorics*, 5(4):371–403, 2000.

[CS96] R. Cleaveland and S. Sims. The ncsu concurrency workbench. In *Proceedings of the 8th conference on Computer Aided Verification (CAV'96)*, number 1102 in Lecture Notes in Computer Science, pages 394–397. Springer, Berlin, 1996.

[DGP97] V. Diekert, P. Gastin, and A. Petit. Removing ϵ–transition in timed automata. In *Proceedings of 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS'97)*, number 1200 in Lecture Notes in Computer Science, pages 583–594. Springer, Berlin, 1997.

[DKL+98] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestré, J. J. Quisquater, and J. L. Willems. A practical implementation of the timing attack. In *Proceedings of CARDIS 1998*, number 1820 in Lecture Notes in Computer Science, pages 167–182. Springer, Berlin, 1998.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In *Proceedings of Hybrid Systems III, Verification and Control 1996*, number 1066 in Lecture Notes in Computer Science. Springer, Berlin, 1996.

[DZ98] F. Demichelis and W. Zielonka. Controlled timed automata. In *Proceedings of 9th International Conference on Concurrency Theory (CONCUR'98)*, number 1466 in Lecture Notes in Computer Science, pages 455–469. Springer, Berlin, 1998.

[EMSS90] E. A. Emerson, A. Mok, A. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proceedings of the 2nd Conference on Computer Aided Verification (CAV'90)*, number 531 in Lecture Notes in Computer Science, pages 136–145. Springer, Berlin, 1990.

[FG96a] Riccardo Focardi and Roberto Gorrieri. Automatic compositional verification of some security properties. In *Proceedings of the 2nd International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'96)*, number 1055 in Lecture Notes in Computer Science, pages 167–186. Springer, Berlin, 1996.

[FG96b] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1996.

[FG97] Riccardo Focardi and Roberto Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.

[FGG97] Riccardo Focardi, A. Ghelli, and Roberto Gorrieri. Using non interference for the analysis of security protocols. In H. Orman and C. Meadows, editors, *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[FGM00] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Information flow in a discrete-time process algebra. In P. Syverson, editor, *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*. IEEE Computer Society Press, 2000.

[FPY02] E. Fersman, P. Petterson, and W. Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *Proceedings of TACAS 2002*, number 2280 in Lecture Notes in Computer Science, pages 67–82. Springer, Berlin, 2002.

[GHJ97] V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In *Proceedings of International Workshop on Hybrid and Real-Time Systems (HART'97)*, number 1201 in Lecture Notes in Computer Science, pages 331–345. Springer, Berlin, 1997.

[GM82] J.A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM Annual Symposium on Principles of Programmig Languages (POPL'80)*, pages 163–173. ACM Press, 1980.

[GS94] Roberto Gorrieri and G. Siliprandi. Real-time system verification using p/t nets. In *Proceedings of 6th Conference on Computer Aided Verification (CAV'94)*, number 818 in Lecture Notes in Computer Science, pages 14–26. Springer, Berlin, 1994.

[HHW97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[HK94] T. A. Henzinger and P. W. Kopke. Verification methods for the divergent runs of clock systems. In *Proceedings of the 2nd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, number 863 in Lecture Notes in Computer Science, pages 351–372. Springer, Berlin, 1994.

[HLP90] H. Harel, O. Lichtenstein, and A. Pnueli. Explicit-clock temporal logic. In *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, pages 402–413. IEEE Computer Society Press, 1990. Cited in ACD93 as an example of the fictitios time model – Introduces the logic XCTL – Cited in AH94 as an example of explicit clock logic.

[HMP94] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112:273–337, 1994. Introduces timed transition systems (every transition has a lower and upper bound of time in which can be fired.

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

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[HW98] P. A. Hsiung and F. Wang. A state-graph manipulator tool for real-time system specification and verification. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA'98)*, 1998.

[JM87] F. Jahanian and A. K. Mok. A graph-theoretic approach for timing analysis and its implementation. *IEEE Transactions on Computers*, 36:961–975, 1987.

[Kel76] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[Kle56] S. C. Kleene. Representations of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–42. Princeton university Press, 1956.

[Koc96] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Proceedings of CRYPTO 1996*, number 1109 in Lecture Notes in Computer Science, pages 104–113. Springer, Berlin, 1996.

[KPSY93] Y. Kesten, A. Pnueli, Joseph Sifakis, and Sergio Yovine. Integration graphs: A class of decidable hybrid systems. In *Hybrid Systems*, number 736 in Lecture Notes in Computer Science. Springer, Berlin, 1993.

[LA92] N. Lynch and H Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6(2):121–139, 1992.

[Lam87] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Programming Languages and Systems*, 5:1–11, 1987.

[Lew90] H. Lewis. A logic of concrete time intervals. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS'90)*, pages 380–389. IEEE Computer Society Press, 1990.

[LMP00] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Adriano Peron. Timed cooperating automata. *Fundamenta Informaticae*, 43:153–173, 2000.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In Horst Reichel, editor, *Proceedings of the 10th International Symposium on Fundamentals of Computation Theory (FCT '95)*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88. Springer, Berlin, 1995.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1(1+2), 1997.

[LS85] N. Leveson and J. Stolzy. Analyzing safety and fault tolerance using timed petri nets. In *Proceedings of the International Joint Conference on*

Theory and Practice of Software Development (TAPSOFT 1985), number 186 in Lecture Notes in Computer Science, pages 339–355. Springer, Berlin, 1985.

[McL94] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of 1994 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1994.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Springer, Berlin, 1980.

[Mul63] D. E. Muller. Infinite sequences and finite machines. In *Proceedings of the 4th annual IEEE Symposium on Swithing Theory and Logical Design*, pages 3–16. IEEE Computer Society Press, 1963.

[NS94] Xavier Nicollin and Joseph Sifakis. The algebra of timed processes atp: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

[Ost90] J. Ostroff. Temporal logic of real-time systems. In *Proceedings of*. Research Studies Press, 1990.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977. Foundations – cited in AH92b as the first attempt of using a temporal logic for specifying reactive systems.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent sistems by petri nets. Technical report, Massachussetts Institute of Technology, 1974.

[RS01] P. Y. A. Ryan and S. A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9(12):75–103, 2001.

[RWW96] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. Non-interference through determinism. *Journal of Computer Security*, 4(1), 1996.

[Sch00] W. Schindler. A timing attack against rsa with the chinese remainder theorem. In *Proceedings of CHES 2000*, number 1965 in Lecture Notes in Computer Science, pages 109–124. Springer, Berlin, 2000. timing-attack.

[SS00] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

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

[WH01] F. Wang and P.-A. Hsiung. Efficient and user-friendly verification. *IEEE Transactions on Computers*, 2001.

[Yov96] Sergio Yovine. Model checking timed automata. In *Lectures on Embedded Systems*, number 1494 in Lecture Notes in Computer Science, pages 114–152. Springer, Berlin, 1996.