Principles of Model Checking
Christel Baier and Joost-Pieter Katoen
MIT Press 2008
Pages 673-698

Chapter 9

Timed Automata

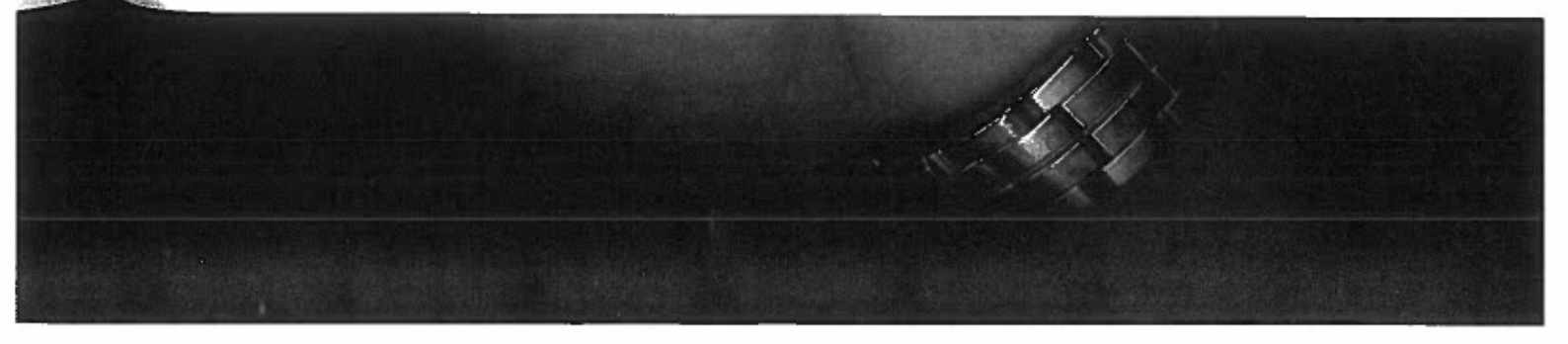
The logics we have encountered so far are interpreted over transition systems that describe how a reactive system may evolve from one state to another. Timing aspects are, however, not covered. That is, indications are given neither about the residence time of a state nor about the possibility of taking a transition within a particular time interval. However, reactive systems such as device drivers, coffee machines, communication protocols, and automatic teller machines, to mention a few, must react in time—they are *time-critical*. The behavior of time-critical systems is typically subject to rather stringent timing constraints. For a train crossing it is essential that on detecting the approach of a train, the gate is closed within a certain time bound in order to halt car and pedestrian traffic before the train reaches the crossing. For a radiation machine the time period during which a cancer patient is subjected to a high dose of radiation is extremely important; a small extension of this period is dangerous and can cause the patient's death.

To put it in a nutshell:

> *Correctness in time-critical systems not only depends on the logical result of the computation but also on the time at which the results are produced.*

As timeliness is of vital importance to reactive systems, it is essential that the timing constraints of the system are guaranteed to be met. Checking whether timing constraints are met is the subject of this chapter. In order to express such timing constraints, the strategy will be to extend logical formalisms that allow expression of the ordering of events, with a notion of quantitative time. Such extensions allow expression of timing constraints such as:

"The traffic light will turn green within the next 30 seconds."



A first choice to be made is the time domain: is it discrete or continuous? A discrete time domain is conceptually simple. Transition systems are used to model timed systems where each action is assumed to last for a single time unit. More general delays can be modeled by using a dedicated unobservable action, τ (for tick), say. The fact that action α lasts $k > 1$ time units may be modeled by $k-1$ tick actions followed (or preceded) by α. This approach typically leads to very large transition systems. Note that in such models the minimal time difference between any pair of actions is a multiple of an a priori, fixed, time unit. For synchronous systems, for instance, in which the involved processes proceed in a lockstep fashion, discrete time domains are appropriate: one time unit corresponds to one clock pulse. In this setting, traditional temporal logics can be used to express timing constraints. The next-step operator can be used to "measure" the discrete elapse of time, i.e., $\bigcirc\Phi$ means that Φ holds after exactly one time unit. By defining $\bigcirc^{k+1}\Phi = \bigcirc^{k}(\bigcirc\Phi)$ and $\bigcirc^{0}\Phi = \Phi$, general timing constraints can be specified. Using the shorthand $\Diamond^{\leqslant k}\Phi = \bigcirc^{0}\Phi \vee \bigcirc\Phi \vee \ldots \vee \bigcirc^{k}\Phi$, the above informally stated timing constraint on the traffic light may be expressed as

$$\Box(red \Rightarrow \Diamond^{\leqslant 30} green).$$

For synchronous systems, transition systems and logics such as LTL or CTL can be used to express timing constraints, and traditional model checking algorithms suffice.

In this monograph we do not want to restrict ourselves to synchronous systems, and will consider—as in Newtonian physics—time of a continuous nature. That is to say, the non-negative real numbers (the set $\mathbb{R}_{\geqslant 0}$) will be used as time domain. A main advantage is that there is no need to fix a minimal time unit in advance as a continuous time model is invariant against changes of the time scale. This is more adequate for asynchronous systems, such as distributed systems, in which components may proceed at distinct speeds, and is more an intuitive than a discrete time model. Transition system representations of asynchronous systems without additional timing information are indeed too abstract to adequately model timing constraints, as illustrated by the following example.

Example 9.1. A Railroad Crossing

Consider a railroad crossing, as discussed in Example 2.30 (page 51), see the schematic representation in Figure 9.1. For this railroad crossing a control system needs to be developed that closes the gate on receipt of a signal indicating that a train is approaching and only opens the gate once the train has signaled that it entirely crossed the road. The safety property that should be established by the control system is that the gates are always closed when the train is crossing the road. The complete system consists of the three components: *Train*, *Gate*, and *Controller*:

$$\mathit{Train} \parallel \mathit{Gate} \parallel \mathit{Controller}.$$

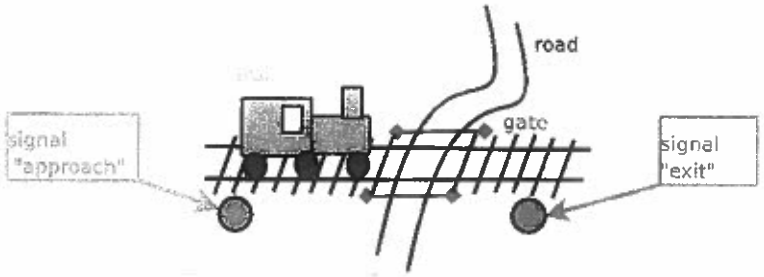


Figure 9.1: Railroad crossing (time abstract).

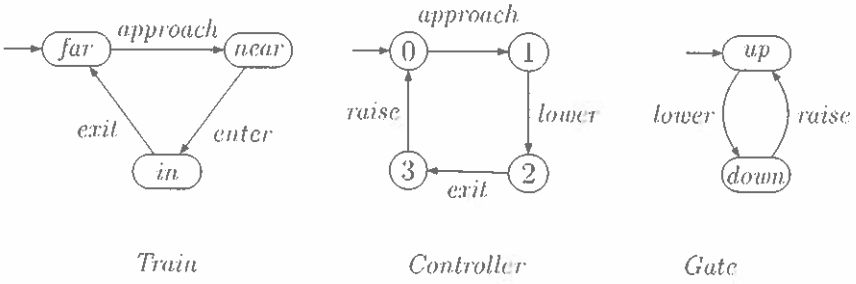


Figure 9.2: Transition systems for processes *Train* (left), *Controller* (middle), and *Gate* (right).

Recall that actions common to a pair of processes need to be performed jointly, while other actions are performed autonomously. The transition systems of these processes are depicted in Figure 9.2. It follows that the composite system $Train \parallel Gate \parallel Controller$ does not guarantee that the gate is closed when the train is passing the crossing. This can easily be seen by inspecting an initial fragment of the composite transition system; see Figure 9.3—it cannot be deduced from the transition system whether, after sending the "approach" signal, the train reaches the road before or after the gate has been closed.

Under the assumption, though, that the train does not exceed a certain maximum speed, a lower bound for the duration between the signal "approach" and the time instant at which the train has reached the crossing can be indicated, see Figure 9.4. Let us assume that the train needs more than 2 minutes to reach the crossing after emission of the "approach" signal. Accordingly, timing assumptions are made for the controller and the gate. On receiving the "approach" signal, after exactly 1 minute the controller will signal the gate to be lowered. The actual closing of the gate is assumed not to exceed a minute. The branching in global state $\langle near, 1, up \rangle$ can now be labeled with timing information:

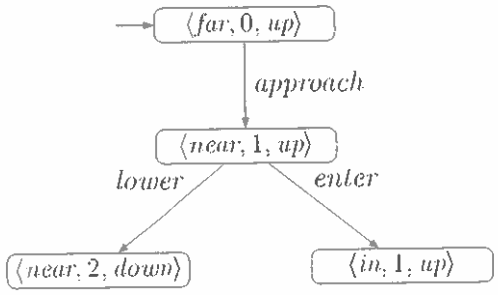


Figure 9.3: Initial fragment of the transition system *Train* ‖ *Controller* ‖ *Gate*.

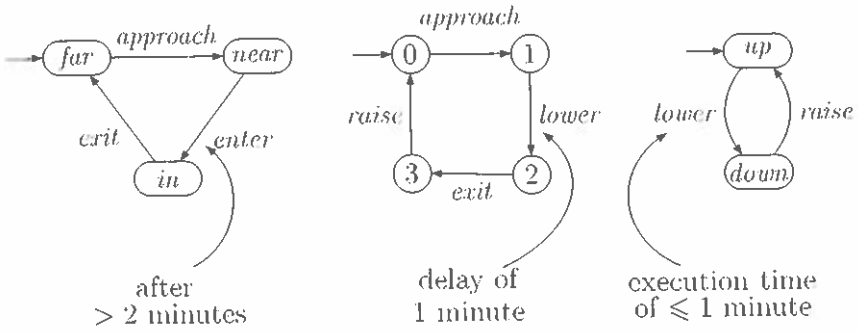
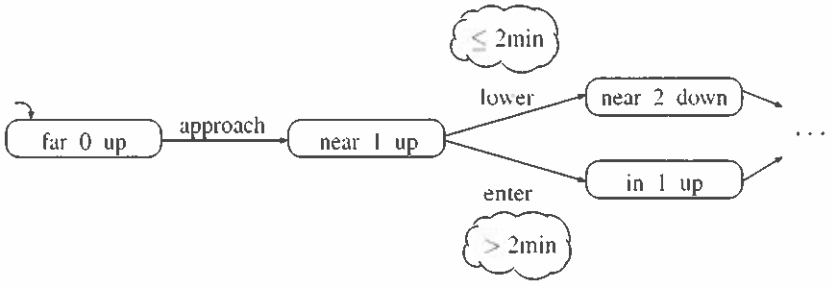


Figure 9.4: The *Train* (left), *Controller* (middle), and *Gate* (right) with timing assumptions.



The train can only execute the local state change $near \xrightarrow{enter} in$ after more than 2 minutes. On the other hand, the gate is closed at most 2 minutes after receiving the "approach" signal. Therefore, the global state change

$$\langle near, 1, up \rangle \xrightarrow{enter} \langle in, 1, up \rangle$$

never occurs. Thus, the gate is always closed before the train reaches the crossing. The fact that the gate remains closed as long as the train is on the crossing is ensured by the fact that action *raise* can only happen after the train has indicated *exit*. ■

As a modeling formalism for time-critical systems, the notion of *timed automata* has been developed, an extension of transition systems (in fact, program graphs) with clock variables that measure the elapse of time. This model includes means to impose constraints on the residence times of states, and on the timing of actions.

9.1 Timed Automata

Timed automata model the behavior of time-critical systems. A timed automaton is in fact a program graph that is equipped with a finite set of real-valued clock variables, called *clocks* for short. In the sequel we assume that the set of clocks is denumerable, and we will use x, y, and z as clocks. Clocks are different from the usual variables, as their access is limited: clocks may only be inspected, and reset to zero. Clocks can be reset to zero after which they start increasing their value implicitly as time progresses. All clocks proceed at rate one, i.e., after the elapse of d time units, all clocks advance by d. The value of a clock thus denotes the amount of time that has been elapsed since its last reset. Clocks can intuitively be considered as stopwatches that can be started and checked independently of one another. Conditions on the values of the clocks are used as enabling conditions (i.e., guards) of actions: only if the condition is fulfilled is the action enabled and capable of being taken; otherwise, the action is disabled. Conditions which depend on clock values are called *clock constraints*. For the sake of simplicity, it is assumed that enabling conditions only depend on clocks and not on other data variables. Clock

constrains are also used to limit the amount of time that may be spent in a location. The following definition prescribes how constraints over clocks are to be defined.

Definition 9.2. Clock Constraint

A *clock constraint* over set C of clocks is formed according to the grammar

$$g ::= \quad x < c \;\Big|\; x \leqslant c \;\Big|\; x > c \;\Big|\; x \geqslant c \;\Big|\; g \wedge g$$

where $c \in \mathbb{N}$ and $x \in C$. Let $CC(C)$ denote the set of clock constraints over C.

Clock constraints that do not contain any conjunctions are *atomic*. Let $ACC(C)$ denote the set of all atomic clock constraints over C.

□

Clock constraints are often written in abbreviated form, e.g., $(x \geqslant c_1) \wedge (x < c_2)$ may be abbreviated by $x \in [c_1, c_2)$ or $c_1 \leqslant x < c_2$. Clock difference constraints such as $x - y < c$ can be added at the expense of a slightly more involved theory. For simplicity, they are omitted here and we restrict the discussion to atomic clock constraints that compare a clock with a constant $c \in \mathbb{N}$. The decidability of the model-checking problem is not affected if c is allowed to be rational. In this case the rationals in each formula can be converted into natural numbers by suitable scaling. In general, we can multiply each constant by the least common multiple of denominators of all constants appearing in all clock constraints.

Intuitively, a timed automaton is a (slightly modified) program graph, whose variables are clocks. The clocks are used to formulate the real-time assumptions on system behavior. An edge in a timed automaton is labeled with a guard (when is it allowed to take an edge?), an action (what is performed when taking the edge?), and a set of clocks (which clocks are to be reset?). A location is equipped with an invariant that constrains the amount of time that may be spent in that location. The formal definition is:

Definition 9.3. Timed Automaton

A *timed automaton* is a tuple $TA = (Loc, Act, C, \hookrightarrow, Loc_0, Inv, AP, L)$ where

- Loc is a finite set of locations,
- $Loc_0 \subseteq Loc$ is a set of initial locations,
- Act is a finite set of actions,
- C is a finite set of clocks,

- $\hookrightarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc$ is a transition relation,
- $Inv : Loc \to CC(C)$ is an invariant-assignment function,
- AP is a finite set of atomic propositions, and
- $L : Loc \to 2^{AP}$ is a labeling function for the locations.

$ACC(TA)$ denotes the set of atomic clock constraints that occur in either a guard or a location invariant of TA. ■

A timed automaton is a program graph with a finite set C of clocks. Edges are labeled with tuples (g, α, D) where g is a clock constraint on the clocks of the timed automaton, α is an action, and $D \subseteq C$ is a set of clocks. The intuitive interpretation of $\ell \overset{g:\alpha,D}{\hookrightarrow} \ell'$ is that the timed automaton can move from location ℓ to location ℓ' when clock constraint g holds. Besides, when moving from location ℓ to ℓ', any clock in D will be reset to zero and action α is performed. Function Inv assigns to each location a location invariant that specifies how long the timed automaton may stay there. For location l, $Inv(\ell)$ constrains the amount of time that may be spent in ℓ. That is to say, the location ℓ should be left before the invariant $Inv(\ell)$ becomes invalid. If this is not possible—as there is no outgoing transition enabled—no further progress is possible. In the formal semantics of timed automata (see Definition 9.11) this situation causes time progress to halt. As time progress is no longer possible, this situation is also known as a *timelock*. This phenomenon will be discussed in more detail later. The function L has the same role as for transition systems and associates to any location the set of atomic propositions that are valid in that location.

Before considering the precise interpretation of timed automata, we give some simple examples.

For depicting timed automata we adopt the drawing conventions for program graphs. Invariants are indicated inside locations and are omitted when they equal true. Edges are labeled with the guard, the action, and the set of clocks to be reset. Empty sets of clocks are often omitted. The same applies to clock constraints that are constantly true. The resetting of set D of clocks is sometimes indicated by $reset(D)$. If the actions are irrelevant, they are omitted.

Example 9.4. Guards vs. Location Invariants

Figure 9.5(a) depicts a simple timed automaton with one clock x and one location ℓ equipped with a self-loop. The self-loop can be taken if clock x has at least the value 2, and when being taken, clock x is reset. Initially, by default clock x has the value 0.

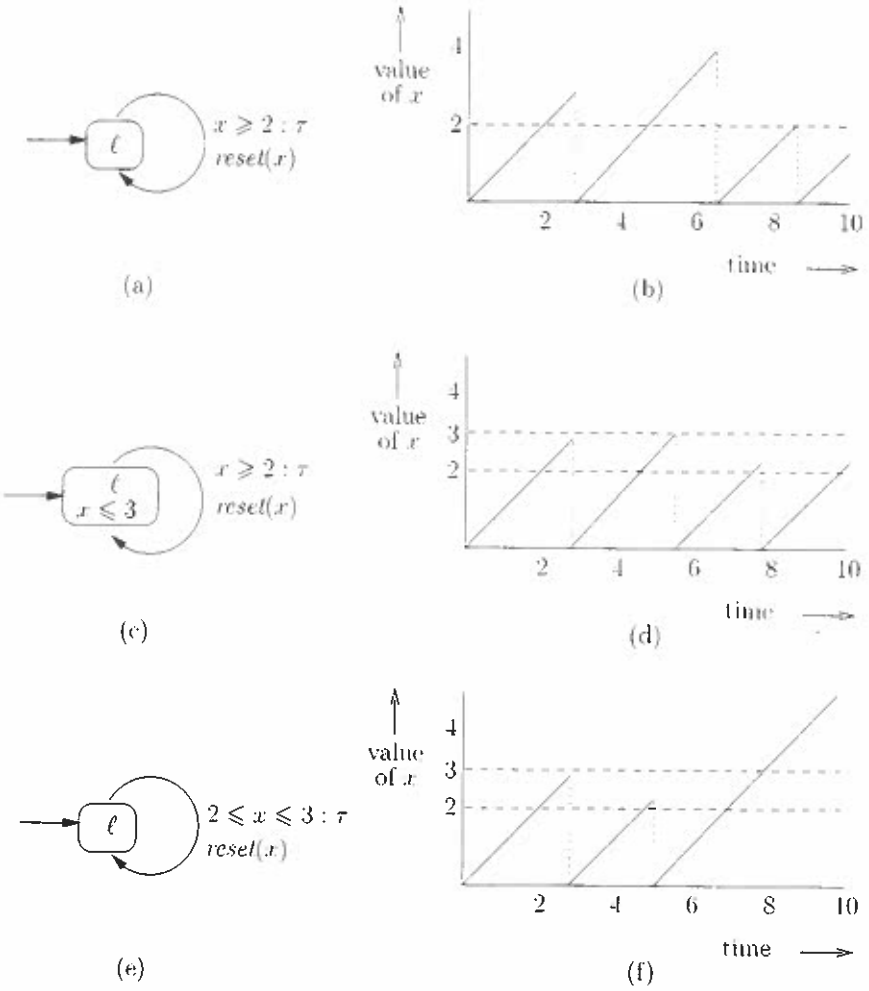


Figure 9.5: Some timed automata with a single clock and one of their evolutions.

Figure 9.5(b) gives an example of an execution of this timed automaton, by depicting the value of clock x vs. the elapsed time since the start of the automaton. Each time the clock is reset to 0, the automaton traverses the self-loop at location ℓ. As $Inv(\ell) = \text{true}$, time can progress without any restriction while residing in ℓ. In particular, a legal behavior of this automaton is to stay in location ℓ ad infinitum. Formally,

$$Loc = Loc_0 = \{\ell\},\ C = \{x\},\ \ell \xrightarrow{\text{true}:x\geqslant 2,\{x\}} \ell, \text{ and } Inv(\ell) = \text{true}.$$

Labelings and actions are omitted.

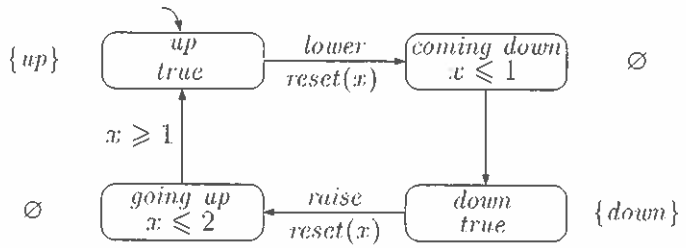


Figure 9.6: Timed automaton for the *Gate*.

Changing the timed automaton of Figure 9.5(a) slightly by incorporating a location invariant $x \leqslant 3$ in location ℓ leads to the effect that x cannot progress unboundedly anymore. Rather, if $x \geqslant 2$ (guard) and $x \leqslant 3$ (invariant) the outgoing transition must be taken. Note that it is not specified at which time instant in the interval $[2, 3]$ the transition is taken, i.e., this is determined nondeterministically. The timed automaton and an example of its behavior are illustrated in Figure 9.5(c) and (d), respectively.

Observe that the same effect is not obtained when strengthening the guard in Figure 9.5(a) into $2 \leqslant x \leqslant 3$ while keeping $Inv(\ell) = \text{true}$. In that case, the outgoing transition can only be taken when $2 \leqslant x \leqslant 3$—as in the previous scenario—but is not forced to be taken, i.e., it can simply be ignored by letting time pass while staying in ℓ. This is illustrated in Figure 9.5(e) and (f).

Put in a nutshell, invariants are the only means to *force* transitions to be taken. ■

Example 9.5. Timed Automaton for the Gate

Consider the gate for the railroad crossing (see Example 9.1, page 674). Assuming that lowering the gate takes at most a single time unit, and raising the gate takes at least one and at most two time units, the timed automaton for process *Gate* is given as in Figure 9.6. We have $Act = \{ lower, raise \}$ and

$$Loc = \{ up, comingdown, down, goingup \}$$

with $Loc_0 = \{ up \}$. The transitions of the timed automaton are

$$up \xrightarrow{\text{true; } lower, \{x\}} comingdown \qquad comingdown \xrightarrow{\text{true; } \tau, \emptyset} down$$

$$down \xrightarrow{\text{true; } raise, \{x\}} goingup \qquad goingup \xrightarrow{x \geqslant 1; \tau, \emptyset} up$$

The location *coming down* with invariant $x \leqslant 1$ has been added to model that the maximal delay between the occurrence of action *lower* and the change to location *down* is at most

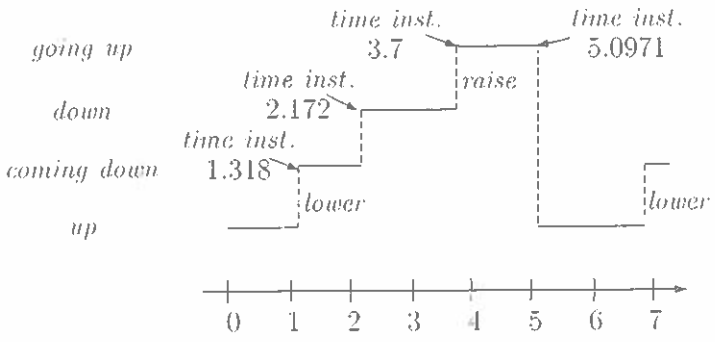


Figure 9.7: Location diagram for the timed automaton *Gate*.

a single time unit. Clock x is set to zero on the occurrence of action *lower* and thus "measures" the elapse of time since that occurrence. By restricting the residence time of *coming down* to $x \leqslant 1$, the switch to *down* must be made within one time unit. Note that this would not have been established by having a direct edge between locations *up* and *down* with guard $x \leqslant 1$, as the value of x would not refer to the time of occurrence of *lower*. In a similar way, the purpose of location *goingup* with invariant $x \leqslant 2$ is to model that raising the gate takes at most two time units. In the initial location *up*, no constraints are imposed on the residence time, i.e., $Inv(up) = \text{true}$. The same applies to location *down*. Let $AP = \{up, down\}$ with the labeling function $L(up) = \{ up \}$, $L(down) = \{ down \}$, and $L(comingdown) = L(goingup) = \varnothing$. ◻

Remark 9.6. Location Diagram

Every finite behavior of a timed automaton can be represented by a *location diagram*. This depicts for every time instant up to some a priori fixed upper bound, the location of the timed automaton during that behavior. For the timed automaton of the *Gate* a possible real-time behavior is indicated by the location diagram in Figure 9.7. ◻

Parallel Composition of Timed Automata For modeling complex systems it is convenient to allow parallel composition of timed automata. This allows for the modeling of time-critical systems in a compositional manner. We consider a parallel composition operator, denoted $||_H$, that is parameterized with a set of *handshaking* actions H. This operator is similar in spirit to the corresponding operator on transition systems; see Definition 2.26, page 48: actions in H need to be performed jointly by both involved timed

automata, whereas actions outside H are performed autonomously in an interleaved fashion.

Definition 9.7. Handshaking for Timed Automata

Let $TA_i = (Loc_i, Act_i, C_i, \hookrightarrow_i, Loc_{0,i}, Inv_i, AP_i, L_i)$, $i = 1, 2$ be timed automata with $H \subseteq Act_1 \cap Act_2$, $C_1 \cap C_2 = \varnothing$ and $AP_1 \cap AP_2 = \varnothing$. The timed automaton $TA_1 \|_H TA_2$ is defined as

$$(Loc_1 \times Loc_2, Act_1 \cup Act_2, C_1 \cup C_2, \hookrightarrow, Loc_{0,1} \times Loc_{0,2}, Inv, AP_1 \cup AP_2, L)$$

where $L(\langle \ell_1, \ell_2 \rangle) = L_1(\ell_1) \cup L_2(\ell_2)$ and $Inv(\langle \ell_1, \ell_2 \rangle) = Inv_1(\ell_1) \wedge Inv_2(\ell_2)$. The transition relation $\hookrightarrow$ is defined by the following rules:

- for $\alpha \in H$:

$$\frac{\ell_1 \xhookrightarrow{g_1 : \alpha, D_1}_1 \ell_1' \ \wedge \ \ell_2 \xhookrightarrow{g_2 : \alpha, D_2}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g_1 \wedge g_2 : \alpha, D_1 \cup D_2} \langle \ell_1', \ell_2' \rangle}$$

- for $\alpha \notin H$:

$$\frac{\ell_1 \xhookrightarrow{g : \alpha, D}_1 \ell_1'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g : \alpha, D} \langle \ell_1', \ell_2 \rangle} \quad \text{and} \quad \frac{\ell_2 \xhookrightarrow{g : \alpha, D}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g : \alpha, D} \langle \ell_1, \ell_2' \rangle}$$

■

The location invariant of a composite location is simply the conjunction of the location invariants of its constituents. For $\alpha \in H$, the transition in the resulting timed automaton is guarded by the conjunction of the guards of the individual timed automata. This entails that an action in H can only be taken when it is enabled in both timed automata. Besides, the clocks that are reset in the individual automata are all reset. As for transition systems, the operator $\|_H$ is associative for a fixed set H. Let $TA_1 \|_H TA_2 \|_H \ldots \| TA_n$ denote the parallel composition of timed automata TA_1 through TA_n where $H \subseteq Act_1 \cap \ldots \cap Act_n$, assuming that all timed automata are compatible, i.e., each pair of TA_i and TA_j, $i \neq j$ have disjoint sets of atomic propositions and disjoint clocks.

Example 9.8. Railroad Crossing

Consider again the railroad crossing example. We extend the timed automaton for the gate (see Example 9.5) with timed automata for the controller and the train. The complete system is then given by

$$(Train \|_{H_1} Controller) \|_{H_2} Gate$$

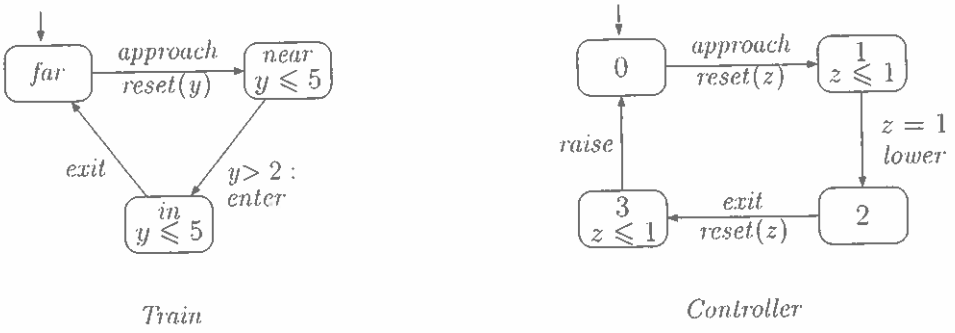


Figure 9.8: Timed automata for the train and the controller .

where $H_1 = \{ approach, exit \}$ and $H_2 = \{ lower, raise \}$.

Let us assume that the train signals its approaching of the gate at least two time units before it enters the railroad crossing. Besides, it is assumed that the train has sufficient speed such that it leaves the crossing five time units after approaching it, at the latest. The timed automaton for the *Train* is depicted in Figure 9.8 (left). On approaching the gate, clock y is set to zero, and only if $y > 2$ is the train allowed to enter the crossing. The *Controller* is depicted in Figure 9.8 (right) and is forced to send the signal "lower" (to the *Gate*) exactly after one time unit after the *Train* has signaled its approaching.

Figure 9.9 shows the composite timed automaton. The prefix of a possible behavior of the complete system is sketched in the location diagram in Figure 9.10.

Note that this timed automaton contains the location $\langle in, 1, up \rangle$. In this location, the train is at the crossing while the gate it still open. However, this location turns out to be unreachable. It can only be reached when $y > 2$, but as y and z are reset at the same time (on entrance of the preceding location), $y > 2$ implies $z > 2$, which is impossible due to the location invariant $z \leqslant 1$. ■

9.1.1 Semantics

The previous examples suggest that the state of a timed automaton is determined by its current location and the current values of all its clocks. In fact, any timed automaton can—like program graphs—be interpreted as a transition system. Due to the continuous time domain, these underlying transition systems have infinitely many states (even uncountably

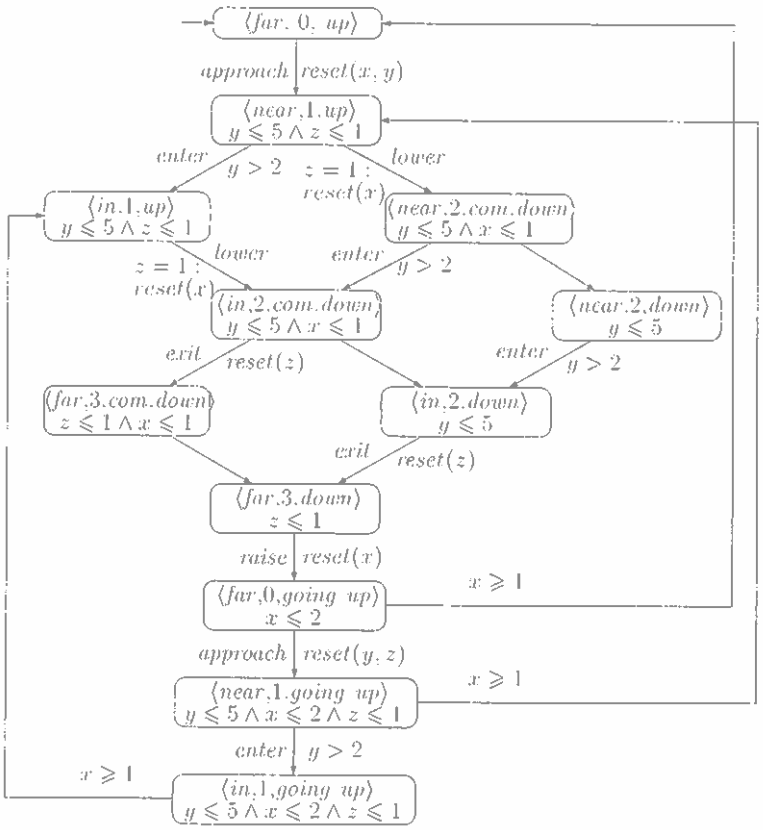


Figure 9.9: The timed automaton $(Train \|_{H_1} Controller) \|_{H_2} Gate$.

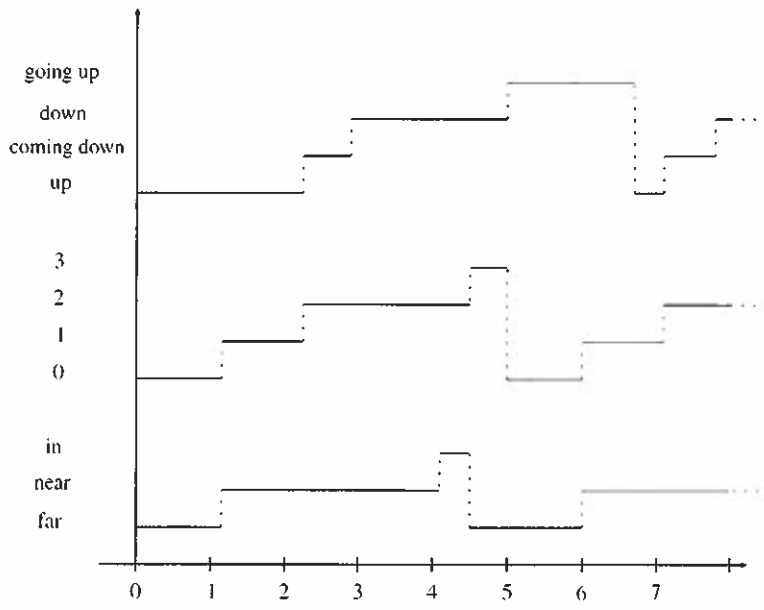


Figure 9.10: Location diagram for a behavior of $(Train \|_{H_1} Controller) \|_{H_2} Gate$.

many), and are infinitely branching. Timed automata can thus be considered as *finite* descriptions of infinite transition systems. The underlying transition system of a timed automaton results from unfolding. Its states consist of a control component, i.e., a location ℓ of the timed automaton, together with a valuation η of the clocks. States are thus pairs of the form $\langle \ell, \eta \rangle$. Let us first consider clock valuations.

Definition 9.9. Clock Valuation

A *clock valuation* η for a set C of clocks is a function $\eta : C \to \mathbb{R}_{\geqslant 0}$, assigning to each clock $x \in C$ its current value $\eta(x)$. □

Let $Eval(C)$ denote the set of all clock valuations over C. In the following, we often use notations like $[x = v, y = v']$ to denote the clock evaluation $\eta \in Eval(\{x, y\})$ with $\eta(x) = v$ and $\eta(y) = v'$.

We can now formally define what it means for a clock constraint to hold for a clock valuation or not. This is done in a similar way as characterizing the semantics of a temporal logic, namely by defining a satisfaction relation. In this case the satisfaction relation $\models$ is a relation between clock valuations (over a set of clocks C) and clock constraints (over C).

Definition 9.10. Satisfaction Relation for Clock Constraints

For set C of clocks, $x \in C$, $\eta \in Eval(C)$, $c \in \mathbb{N}$, and $g, g' \in CC(C)$, let $\models \subseteq Eval(C) \times CC(C)$ be defined by

$$\begin{array}{ll} \eta \models \text{true} & \\ \eta \models x < c & \text{iff } \eta(x) < c \\ \eta \models x \leqslant c & \text{iff } \eta(x) \leqslant c \\ \eta \models \neg g & \text{iff } \eta \not\models g \\ \eta \models g \wedge g' & \text{iff } \eta \models g \ \wedge \ \eta \models g' \end{array}$$

■

Let η be a clock valuation on C. For positive real d, $\eta{+}d$ denotes the clock valuation where all clocks of η are increased by d. Formally, $(\eta{+}d)(x) = \eta(x) + d$ for all clocks $x \in C$. reset x in η denotes the clock valuation which is equal to η except that clock x reset. Formally:

$$(\text{reset } x \text{ in } \eta)(y) = \begin{cases} \eta(y) & \text{if } y \neq x \\ 0 & \text{if } y = x. \end{cases}$$

For the clock valuation $\eta = [x = \pi, y = 4]$, valuation $\eta{+}9 = [x = \pi{+}9, y = 13]$, and reset x in $(\eta{+}9) = [x = 0, y = 13]$. Nested occurrences of reset are typically abbreviated. For instance, reset x in (reset y in η) is denoted reset x, y in η.

There are two possible ways in which a timed automaton can proceed: by taking a transition in the timed automaton, or by letting time progress while staying in a location. In the underlying transition system, the former is represented by a *discrete* transition and the latter by a *delay* transition. In the former case, the corresponding transition of the underlying transition system is labeled with the action of the transition in the timed automaton, in the latter case, it is labeled with a positive real number indicating the amount of time that has elapsed.

Definition 9.11. Transition System Semantics of a Timed Automaton

Let $TA = (Loc, Act, C, \hookrightarrow, Loc_0, Inv, AP, L)$ be a timed automaton. The transition system $TS(TA) = (S, Act', \rightarrow, I, AP', L)$ with:

- $S = Loc \times Eval(C)$
- $Act' = Act \cup \mathbb{R}_{\geqslant 0}$
- $I = \{ \langle \ell_0, \eta \rangle \mid \ell_0 \in Loc_0 \ \wedge \ \eta(x) = 0 \text{ for all } x \in C \}$
- $AP' = AP \cup ACC(C)$

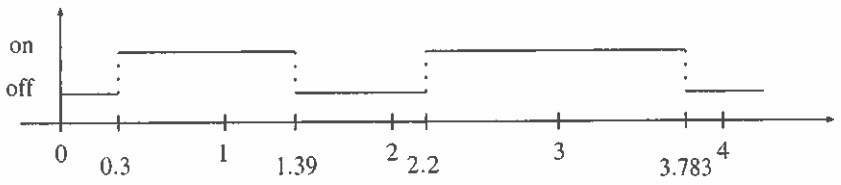
- $L'(\langle \ell, \eta \rangle) = L(\ell) \cup \{ g \in ACC(C) \mid \eta \models g \}$
- the transition relation $\longrightarrow$ is defined by the following two rules:
  - *discrete transition*: $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \eta' \rangle$ if the following conditions hold:
    (a) there is a transition $\ell \overset{g:\alpha,D}{\hookrightarrow} \ell'$ in TA
    (b) $\eta \models g$
    (c) $\eta' = \mathsf{reset}\ D\ \mathsf{in}\ \eta$
    (d) $\eta' \models Inv(\ell')$
  - *delay transition*: $\langle \ell, \eta \rangle \xrightarrow{d} \langle \ell, \eta+d \rangle$ for $d \in \mathbb{R}_{\geqslant 0}$
    (e) if $\eta+d \models Inv(\ell)$

■

For a transition that corresponds to (a) traversing a transition $\ell \overset{g:\alpha,D}{\hookrightarrow} \ell'$ in the timed automaton TA it must hold that (b) η satisfies the clock constraint g (ensuring the transition is enabled), and (c) the new clock valuation η' is obtained by resetting all clocks D in η should (d) satisfy the location invariant of ℓ' (otherwise it is not allowed to be in ℓ'). Idling in a location (second clause) for some non-negative amount of time is allowed (e) if the location invariant remains true while time progresses. For state $\langle \ell, \eta \rangle$ such that $\eta \models Inv(\ell)$, there are typically uncountably many outgoing delay transitions of the form $\langle \ell, \eta \rangle \xrightarrow{d}$ as d can be selected from a continuous domain.

Example 9.12. Light Switch

The timed automaton *Switch* in Figure 9.11 models a switch that controls a light. When off, the switch may be turned on at any time instant. The user may switch off the light at least one time unit after the most recent time the light was switched on. After two time units the light automatically switches off. Clock x is used to keep track of the delay since the last time the light has been switched on. (The timed automaton does not distinguish between the *switch_off* action activated by the user and by the light. This could be made explicit by adding an edge from location *on* to *off*, with guard $x = 2$ and action τ.) The following location diagram indicates a possible behavior:



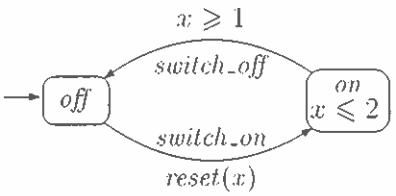


Figure 9.11: A simple light switch.

The transition system $TS(Switch)$ has the state space

$$S \;=\; \{\langle off, t\rangle \mid t \in \mathbb{R}_{\geqslant 0}\} \;\cup\; \{\langle on, t\rangle \mid t \in \mathbb{R}_{\geqslant 0}\}$$

where t is a shorthand for the clock evaluation η with $\eta(x) = t$. Uncountably many transitions emanate from the initial state $\langle off, 0\rangle$. $TS(Switch)$ has the following transitions for reals d and t:

$$\begin{array}{lcll}
\langle off, t\rangle & \xrightarrow{d} & \langle off, t+d\rangle & \text{for all } t \geqslant 0 \text{ and } d \geqslant 0 \\
\langle off, t\rangle & \xrightarrow{switch_on} & \langle on, 0\rangle & \text{for all } t \geqslant 0 \\
\langle on, t\rangle & \xrightarrow{d} & \langle on, t+d\rangle & \text{for all } t \geqslant 0 \text{ and } d \geqslant 0 \text{ with } t+d \leqslant 2 \\
\langle on, t\rangle & \xrightarrow{switch_off} & \langle off, t\rangle & \text{for all } 1 \leqslant t \leqslant 2.
\end{array}$$

The set of reachable states in $TS(Switch)$ from state $\langle off, 0\rangle$ is

$$\{\langle off, t\rangle \mid t \in \mathbb{R}_{\geqslant 0}\} \;\cup\; \{\langle on, t\rangle \mid 0 \leqslant t \leqslant 2\}$$

as the location invariant $x \leqslant 2$ is violated in any state $\langle on, t\rangle$ with $t > 2$. A prefix of an example path of $TS(Switch)$ is

$$\langle off, 0\rangle \xrightarrow{0.57} \langle off, 0.57\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{\sqrt{2}} \langle on, \sqrt{2}\rangle \xrightarrow{0.2}$$
$$\langle on, \sqrt{2}{+}0.2\rangle \xrightarrow{switch\_off} \langle off, \sqrt{2}{+}0.2\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{1.7} \langle on, 1.7\rangle \ldots$$

■

Remark 9.13. Parallel Composition

For timed automata we have

$$TS(TA_1) \parallel_{H \cup \mathbb{R}_{>0}} TS(TA_2) \;=\; TS(TA_1 \parallel_H TA_2)$$

up to isomorphism. This is due to the fact that TA_1 and TA_2 do not have any shared variables. Synchronization over time passage actions reflects the natural fact that time proceeds equally fast in both components. ■

The paths in $TS(TA)$ are discrete representations of the continuous-time "behavior" of TA. They indicate at least the states immediately before and after the execution of an action $\alpha \in Act$. However, due to the fact that, e.g., interval delays may be realized in uncountably many ways, different paths may describe the same behavior (i.e., location diagram). Consider, e.g., the behavior of the light switch of Example 9.12 where the light alternates between *off* and *on* while being *off* for exactly one time unit and *on* for two time units, i.e., a return to *off* takes place after exactly three time units. The following three example paths correspond to this continuous-time behavior:

$$\begin{array}{lllllllllll}
\pi_1 = & \langle off,0\rangle & & \langle off,1\rangle & \langle on,0\rangle & & & \langle on,2\rangle & \langle off,2\rangle & \cdots \\
\pi_2 = & \langle off,0\rangle & \langle off,0.5\rangle & \langle off,1\rangle & \langle on,0\rangle & & \langle on,1\rangle & \langle on,2\rangle & \langle off,2\rangle & \cdots \\
\pi_3 = & \langle off,0\rangle & \langle off,0.1\rangle & \langle off,1\rangle & \langle on,0\rangle & \langle on,0.53\rangle & \langle on,1.3\rangle & \langle on,2\rangle & \langle off,2\rangle & \cdots
\end{array}$$

The only difference between these paths is the delay transitions. In path π_1, the one time unit residence in the initial state $\langle off,0\rangle$ is realized by means of the delay transition $\langle off,0\rangle \xrightarrow{1} \langle off,1\rangle$. In contrast, the paths π_2 and π_3 realize this single time unit by two delay transitions:

$$\langle off,0\rangle \xrightarrow{0.5} \langle off,0.5\rangle \xrightarrow{0.5} \langle off,1\rangle \quad \text{and}$$
$$\langle off,0\rangle \xrightarrow{0.1} \langle off,0.1\rangle \xrightarrow{0.9} \langle off,1\rangle.$$

But the effect of the transition $\langle \ell,\eta\rangle \xrightarrow{d_1+d_2} \langle \ell,\eta+d_1+d_2\rangle$ corresponds to the effect of the sequence of transitions:

$$\langle \ell,\eta\rangle \xrightarrow{d_1} \langle \ell,\eta+d_1\rangle \xrightarrow{d_2} \langle \ell,\eta+d_1+d_2\rangle.$$

In both cases, d_1+d_2 time units pass without executing an action $\alpha \in Act$. Thereby, uncountably many states of the form $\langle \ell,\eta+t\rangle$ with $0 \leqslant t \leqslant d_1+d_2$ are passed through.

Remark 9.14. Multiple Actions in Zero Time

The elapse of time in timed automata only takes place in locations. Actions $\alpha \in Act$ take place instantaneously, i.e., they have a duration of zero time units. As a result, at a single time instant, several actions take place. ■

9.1.2 Time Divergence, Timelock, and Zenoness

The semantics of a timed automaton is given by a transition system with uncountably many states (and transitions). Paths through this transition system correspond to possible

behaviors of the timed automaton. However, not every such path represents a realistic behavior. This subsection treats three essential phenomena for timed automata: time divergence, timelock, and zenoness.

Time Divergence Consider a location ℓ such that for any $t < d$, for fixed constant $d \in \mathbb{R}_{>0}$, clock valuation $\eta + t \models \mathit{Inv}(\ell)$. A possible execution fragment starting from this location is

$$\langle \ell, \eta \rangle \xrightarrow{d_1} \langle \ell, \eta + d_1 \rangle \xrightarrow{d_2} \langle \ell, \eta + d_1 + d_2 \rangle \xrightarrow{d_3} \langle \ell, \eta + d_1 + d_2 + d_3 \rangle \xrightarrow{d_4} \ldots$$

where $d_i > 0$ and the infinite sequence $d_1 + d_2 + \ldots$ converges toward d. Such infinite path fragments are called *time-convergent*. A time-convergent path is counterintuitive as time advances only up to a certain value whereas by nature time always progresses. For example, the transition system of the timed automaton for the light switch (see Figure 9.11 on page 689) exhibits the time-convergent execution fragment

$$\langle \mathit{off}, 0 \rangle \xrightarrow{2^{-1}} \langle \mathit{off}, 1-2^{-1} \rangle \xrightarrow{2^{-2}} \langle \mathit{off}, 1-2^{-2} \rangle \xrightarrow{2^{-3}} \langle \mathit{off}, 1-2^{-3} \rangle \ldots\ldots$$

which visits infinitely many states in the interval $[\frac{1}{2}, 1]$. Time never proceeds beyond one. The corresponding path is time-convergent. As time-convergent paths are not realistic, they are not considered. That is to say, the analysis of timed automata is focused on *time-divergent* paths, i.e., paths in which time always progresses.

In order to formally define time-divergent paths, let us first define the elapsed time of a path. Intuitively, the elapsed time of a path is the total time that elapses along a path. The duration of an action $\alpha \in \mathit{Act}$ is zero; the duration of a delay action d is d.

Definition 9.15. Elapsed Time on a Path

Let TA be a timed automaton with the set Act of actions. The function $\mathit{ExecTime} : \mathit{Act} \cup \mathbb{R}_{>0} \to \mathbb{R}_{\geqslant 0}$ is defined as

$$\mathit{ExecTime}(\tau) = \begin{cases} 0 & \text{if } \tau \in \mathit{Act} \\ d & \text{if } \tau = d \in \mathbb{R}_{>0}. \end{cases}$$

For infinite execution fragment $\rho = s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \ldots$ in $TS(TA)$ with $\tau_i \in \mathit{Act} \cup \mathbb{R}_{>0}$ let

$$\mathit{ExecTime}(\rho) = \sum_{i=0}^{\infty} \mathit{ExecTime}(\tau_i).$$

The execution time of finite execution fragments is defined analogously. For the path fragment π in $TS(TA)$ induced by ρ, $\mathit{ExecTime}(\pi) = \mathit{ExecTime}(\rho)$. ∎

Note that path fragment π may be induced by several execution fragments. However, every pair of execution fragments with the same path fragment are only distinguished by discrete transitions and not by delay transitions. Hence, $ExecTime(\pi)$ is well-defined.

Definition 9.16. Time Divergence and Time Convergence

The infinite path fragment π is *time-divergent* if $ExecTime(\pi) = \infty$; otherwise, π is *time-convergent*.

Example 9.17. Light Switch

For the light switch described in Example 9.12 (page 688), the path π in $TS(Switch)$ in which on and off periods of 1 minute alternate:

$$\pi = \langle off, 0\rangle \langle off, 1\rangle \langle on, 0\rangle \langle on, 1\rangle \langle off, 1\rangle \langle off, 2\rangle \langle on, 0\rangle \langle on, 1\rangle \langle off, 1\rangle \ldots$$

is time-divergent as $ExecTime(\pi) = 1 + 1 + 1 + \ldots = \infty$. The path

$$\pi' = \langle off, 0\rangle \langle off, 1/2\rangle \langle off, 3/4\rangle \langle off, 7/8\rangle \langle off, 15/16\rangle \ldots$$

in $TS(Switch)$ is time-convergent, as $ExecTime(\pi') = \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^{i+1} = 1 < \infty$.

Definition 9.18. Time-Divergent Set of Paths

For state s in $TS(TA)$ let: $Paths_{div}(s) = \{\, \pi \in Paths(s) \mid \pi \text{ is time-divergent} \,\}$.

That is, $Paths_{div}(s)$ denotes the set of time-divergent paths in $TS(TA)$ that start in s. Although time-convergent paths are not realistic, their existence cannot be avoided. For the analysis of timed automata, time-convergent paths are simply ignored, e.g., a timed automaton satisfies an invariant when along all its time-divergent paths the invariant is satisfied.

Timelock. State s in $TS(TA)$ contains a *timelock* if there is no time-divergent path starting in s. Such states are unrealistic as time cannot progress for ever from these states. Timelocks are considered as undesired and need to be avoided when modeling a time-critical system by means of a timed automaton.

Definition 9.19. Timelock

Let TA be a timed automaton. State s in $TS(TA)$ contains a *timelock* if $Paths_{div}(s) = \varnothing$. TA is *timelock-free* if no state in $Reach(TS(TA))$ contains a timelock.

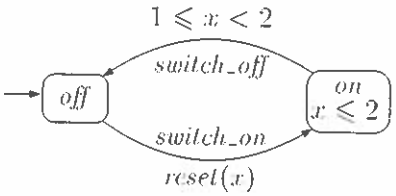


Figure 9.12: Timed automaton $Switch_1$.

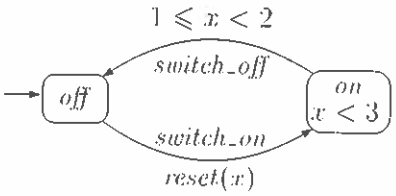


Figure 9.13: Timed automaton $Switch_2$.

Example 9.20. Modified Light Switch

We modify the light switch such that the light is on for a period with duration $t \in [1, 2)$, i.e., the light is always switched off within 2 minutes; see the timed automaton $Switch_1$ in Figure 9.12. The state $\langle on, 2 \rangle$ is reachable in transition system $TS(Switch_1)$, e.g., via the execution fragment:

$$\langle off, 0 \rangle \xrightarrow{switch\_on} \langle on, 0 \rangle \xrightarrow{2} \langle on, 2 \rangle.$$

As $\langle on, 2 \rangle$ is a terminal state, $Paths_{div}(\langle on, 2 \rangle) = \varnothing$, and the state contains a timelock. Timed automaton $Switch_1$ is thus not timelock-free.

Any terminal state of a transition system that results from a timed automaton contains a timelock. Terminal states should not be confused with terminal locations, i.e., locations that have no outgoing edges. A terminal location ℓ with $Inv(\ell) = \text{true}$, e.g., does not result in a terminal state in the underlying transition system, as time may progress in ℓ for ever. Terminal locations thus do not necessarily yield states with timelocks.

Not only terminal states may contain timelocks. Consider, e.g., another variant of the light switch where $Inv(on) = x < 3$, see timed automaton $Switch_2$ in Figure 9.13. The reachable state $\langle on, 2 \rangle$ is not terminal, , e.g., the time-convergent path in $TS(Switch_2)$:

$$\langle on, 2 \rangle \langle on, 2.9 \rangle \langle on, 2.99 \rangle \langle on, 2.999 \rangle \langle on, 2.9999 \rangle \ldots$$

emanates from it. However, $Paths_{div}(\langle on, 2 \rangle) = \varnothing$ as the state $\langle on, 2 \rangle$ has no outgoing discrete transitions (as the guard $1 \leqslant x < 2$ is violated), and time cannot progress beyond three (due to $Inv(on) = x < 3$). State $\langle on, 2 \rangle$ in $TS(Switch_2)$ contains a timelock. Timed automaton $Switch_2$ is thus not timelock-free. ■

Zenoness As opposed to the presence of time-convergent paths, timelocks are considered as modeling flaws that should be avoided. The latter also applies to zenoness. Recall that the execution of actions $\alpha \in Act$ is instantaneous, i.e., actions take no time. Without further restrictions, a timed automaton may perform infinitely many actions in a finite time interval. This phenomenon is also called *zeno* and represents nonrealizable behavior, since it would require infinitely fast processors.

Definition 9.21. Zeno Paths

Let TA be a timed automaton. The infinite path fragment π in $TS(TA)$ is *zeno* (or: a *zeno path*) if it is time-convergent and infinitely many actions $\alpha \in Act$ are executed along π. ◻

Definition 9.22. Nonzenoness

A timed automaton TA is *non-zeno* if there does not exist an initial zeno path in $TS(TA)$. ◻

Timed automaton TA is thus non-zeno if and only if for every path π in $TS(TA)$, either π is time-divergent or π is time-convergent with almost only (i.e., all except for finitely many) delay transitions.

Note that non-zenoness, as well as timelock freedom, only refers to the *reachable* fragment of the transition system $TS(TA)$. A non-zeno timed automaton may possess zeno paths starting in an unreachable state. Similarly, a timelock-free timed automaton may contain unreachable timelock states.

Example 9.23. Zeno Paths of a Light Switch

Consider yet another variant of the light switch, in which the user has the possibility to push the on button while the light is on. While doing so, clock x is reset, and the light stays on for at most two units, unless the user pushes the on button again: see timed automaton $Switch_3$ in Figure 9.14. The paths induced by the following execution fragments of $TS(Switch_3)$

$$\langle off, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{switch\_on} \ldots$$

$$\langle off, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{0.5} \langle on, 0.5\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{0.25} \langle on, 0.25\rangle \xrightarrow{switch\_on} \ldots,$$

are zeno paths during which the user presses the on button infinitely fast, or faster and faster, respectively.

This unrealizable behavior can be avoided by imposing a minimal non-zero delay, c, say, between successive button pushings by the user. This is established by imposing $Inv(on) =$

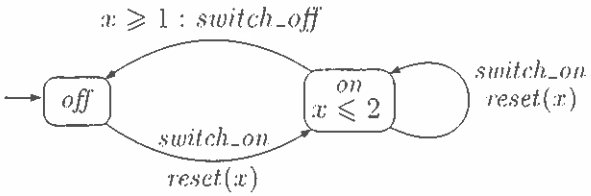


Figure 9.14: Timed automaton $Switch_3$.

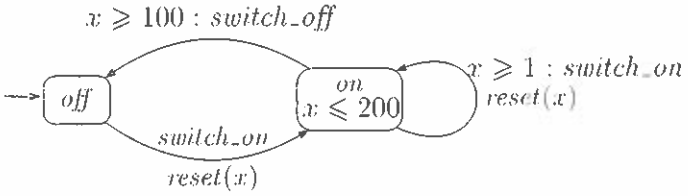


Figure 9.15: Timed automaton $Switch_4$.

$x \geqslant c$ for $c > 0$. Note that c should be a natural number. In order to model a minimal response $0 < c < 1$ where c is rational, say $c = \frac{1}{100}$, all time constraints in the timed automaton $Switch_3$ need to be rescaled (see Figure 9.15). Essentially, the timed automaton $Switch_4$ computes with a modified time unit: one time unit for x in $Switch_4$ corresponds to $\frac{1}{100}$ minutes.

■

To check whether or not a timed automaton is non-zeno is algorithmically difficult. Instead, sufficient conditions are considered that are simple to check, e.g., by a static analysis of the timed automaton. The following criterion is based on the intuition that a timed automaton is non-zeno if on any of its control cycles, time advances with at least some constant amount (larger than zero). This yields:

Lemma 9.24. Sufficient Criterion for Nonzenoness

Let TA be a timed automaton with set C of clocks such that for every control cycle in TA

$$\ell_0 \xrightarrow{g_1:a_1,C_1} \ell_1 \xrightarrow{g_2:a_2,C_2} \ldots \xrightarrow{g_n:a_n,C_n} \ell_n \text{ with } \ell_0 = \ell_n.$$

there exists a clock $x \in C$ such that

1. *$x \in C_i$ for some $0 < i \leqslant n$, and*

2. *for all clock evaluations* η *there exists* $c \in \mathbb{N}_{>0}$ *such that*

$$\eta(x) < c \quad \textit{implies} \quad (\eta \not\models g_j \textit{ or } \mathit{Inv}(\ell_j)), \textit{ for some } 0 < j \leqslant n.$$

Then: TA *is non-zeno.*

Proof: Let TA be a timed automaton over C with $x \in C$ satisfying the two constraints stated in the claim and i, j the corresponding indices. Let π be a path in $TS(TA)$ that performs infinitely many actions $\alpha \in Act$. As TA contains finitely many states, π traverses some control cycle $\ell_0 \hookrightarrow \ell_1 \hookrightarrow \ldots \hookrightarrow \ell_n = \ell_0$. Assume that $i \leqslant j$. (As locations on cycles can be renumbered, this is not a restriction.) Consider the path fragment in $TS(TA)$ that starts and ends in location ℓ_0:

$$\langle \ell_0, \eta_0 \rangle \ldots \langle \ell_{i-1}, \eta_{i-1} \rangle \langle \ell_i, \eta_i \rangle \ldots \langle \ell_{j-1}, \eta_{j-1} \rangle \langle \ell_j, \eta_j \rangle \ldots \langle \ell_0, \eta'_0 \rangle.$$

By the constraints satisfied by TA, clock x is reset on the transition $\ell_{i-1} \hookrightarrow \ell_i$, and the transition $\ell_{j-1} \hookrightarrow \ell_j$ is only possible when $\eta_{i-1}(x) \geqslant c$ (as for $\eta_{i-1}(x) < c$, either the guard or the location invariant of ℓ_j are violated). This implies that by traversing the cycle $\ell_0 \hookrightarrow \ldots \hookrightarrow \ell_0$ time advances with at least $c > 0$ time units. Hence, π is time-divergent and TA is non-zeno. ∎

The condition in Lemma 9.24 is compositional, i.e., if TA and TA' both satisfy the constraints, then the parallel composed timed automaton $TA \parallel TA'$ also satisfies the constraints. This follows directly from the fact that a control cycle in $TA \parallel TA'$ consists of a control cycle in TA, or TA', or both. If each control cycle in TA and in TA' satisfies the constraint that time should advance with at least some positive amount, then this thus also applies to each control cycle in $TA \parallel TA'$. This property significantly simplifies to checking whether a composite timed automaton is non-zeno. In case a timed automaton is in fact untimed (as no clocks are used or all guards and invariants are vacuously true), it can be considered as non-zeno, and thus not affect the control cycles of other component timed automata in a composite system.

Example 9.25. Sufficient Condition for Nonzenoness

The timed automaton in Figure 9.15 (page 695) satisfies the constraints in Lemma 9.24. In the control cycle *off* $\hookrightarrow$ *on* $\hookrightarrow$ *off*, clock x is reset on *off* $\hookrightarrow$ *on*, and the guard $x \geqslant 100$ ensures that when going from location *off* to *on*, time has advanced with at least 100 time units. On the control cycle *on* $\hookrightarrow$ *on*, clock x is reset, and the guard $x \geqslant 1$ ensures that at least one time unit elapses on traversing this control cycle.

The timed automata *Train*, *Gate*, and *Controller* of Example 9.8 (page 683) all satisfy for any control cycle the constraints in Lemma 9.24. This can be seen as follows. Timed

automaton *Gate* has one control cycle: $up \hookrightarrow \ldots \hookrightarrow up$. In that cycle, clock x is reset when moving from location *down* to *going_up*. Furthermore, for $\eta(x) < 1$, location *up* is not reachable due to the guard $x \geqslant 1$ on *going_up* $\hookrightarrow$ *up*. This ensures that on traversing the control cycle $up \hookrightarrow \ldots \hookrightarrow up$, time advances with at least one time unit. The timed automaton *Train* contains the control cycle $far \hookrightarrow \ldots \hookrightarrow far$. However, clock y is reset on that cycle before reaching location *near*, and the guard $y > 2$ on *near* $\hookrightarrow$ *in* guarantees that on traversing the control cycle at least one (in fact, more than two) time unit has elapsed. For *Controller*, the resetting of clock z and guard $z{=}1$ ensure that also this timed automaton fulfills the constraints of Lemma 9.24. The timed automata *Train*, *Gate* and *Controller* are thus non-zeno. As the control cycles in the composed timed automaton $(Train \parallel_{H_1} Gate) \parallel_{H_2} Controller$ are comprised of the constituting timed automata, this composed timed automaton is non-zeno.

■

The previous considerations indicate that a timed automaton is adequately modeling a time-critical system whenever it is non-zeno and does not contain any timelock. Timelock-free, non-zeno timed automata induce transition systems without terminal states such that along any path only finitely many actions are executed in finite time. In contrast to zeno paths and timelocks, time-convergent paths will be treated akin to unfair paths (in fair CTL) and are explicitly excluded for analysis purposes.

A delay of $d > 0$ time units can be realized in different ways, in general by $n > 0$ delay transitions of size d_1 through d_n with $d_i > 0$ such that $d = d_1 + \ldots + d_n$. As we are only interested in the amount of time advancing, a sequence of delay transitions labeled with d_1 through d_n and a sequence labeled with d'_1 through d'_k, say, such that $\sum_{i=1}^{n} d_i = \sum_{i=1}^{k} d'_i = d$, are considered equivalent and denoted by $\stackrel{d}{\Longrightarrow}$. This relation is used later for defining the semantics of timed CTL.

Notation 9.26. Sets of Path Fragments

Let *TA* be a timed automaton. For path fragments in *TS*(*TA*) along which infinitely many actions are performed, let

$$s_0 \stackrel{d_0}{\Longrightarrow} s_1 \stackrel{d_1}{\Longrightarrow} s_2 \stackrel{d_2}{\Longrightarrow} \ldots \qquad \text{with } d_0, d_1, d_2 \ldots \geqslant 0$$

denote the equivalence class containing all infinite path fragments induced by execution fragments in *TS*(*TA*) of the form

$$s_0 \underbrace{\stackrel{d_0^1}{\rightarrow} \ldots \stackrel{d_0^{k_0}}{\rightarrow}}_{\substack{\text{time passage of} \\ d_0 \text{ time-units}}} s_0 + d_0 \stackrel{a_0}{\longrightarrow} s_1 \underbrace{\stackrel{d_1^1}{\rightarrow} \ldots \stackrel{d_1^{k_1}}{\rightarrow}}_{\substack{\text{time passage of} \\ d_1 \text{ time-units}}} s_1 + d_1 \stackrel{a_1}{\longrightarrow} s_2 \underbrace{\stackrel{d_2^1}{\rightarrow} \ldots \stackrel{d_2^{k_2}}{\rightarrow}}_{\substack{\text{time passage of} \\ d_2 \text{ time-units}}} s_2 + d_2 \stackrel{a_2}{\longrightarrow} \ldots$$

where $k_i \in \mathbb{N}$, $d_i \in \mathbb{R}_{\geqslant 0}$ and $\alpha_i \in Act$ such that $\sum_{j=1}^{k_i} d_i^j = d_i$. Note that in the $\Longrightarrow$ notations, actions are abstracted away.

For infinite path fragment $\pi \in s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \ldots$ that performs infinitely many actions, we have $ExecTime(\pi) = \sum_{i \geqslant 0} d_i$. Path fragments in $s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \ldots$ are time-divergent if and only if $\sum_i d_i$ diverges.

Time-divergent path fragments that perform finitely many actions $\alpha \in Act$ (but contain infinitely many delay transitions) are represented in a similar way, except that after the execution of the last action $\alpha \in Act$ the advance of time is represented by infinitely many $\overset{1}{\Longrightarrow}$ transitions. That is, the set

$$s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \ldots \overset{d_{n-1}}{\Longrightarrow} s_n \overset{1}{\Longrightarrow} s_n{+}1 \overset{1}{\Longrightarrow} s_n{+}2 \overset{1}{\Longrightarrow} \ldots$$

contains all infinite path fragments induced by the execution fragment of the form

$$s_0 \underbrace{\rightarrow \ldots \rightarrow}_{\text{time passage of } d_0 \text{ time-units}} \xrightarrow{\alpha_0} \ldots \xrightarrow{\alpha_{n-2}} s_{n-1} \underbrace{\rightarrow \ldots \rightarrow}_{\text{time passage of } d_{n-1} \text{ time-units}} \xrightarrow{\alpha_{n-1}} s_n \underbrace{\xrightarrow{1} s_n{+}1 \xrightarrow{1} s_n{+}2 \xrightarrow{1} \ldots}_{\text{infinite time passage}}$$

Hence, $s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \ldots$ is a uniform notation for all infinite time-divergent path fragments. ■

9.2 Timed Computation Tree Logic

Timed CTL (TCTL, for short) is a real-time variant of CTL aimed to express properties of timed automata. In TCTL, the until modality is equipped with a time interval such that $\Phi \, \mathsf{U}^{J} \, \Psi$ asserts that a Ψ-state is reached within $t \in J$ time units while only visiting Φ-states before reaching the Ψ-state. The fact that a deadlock may be reached within thirty time units via legal states only, say, can be expressed as $\mathit{legal} \, \mathsf{U}^{[0,30]} \, \mathit{deadlock}$, where the atomic propositions *legal* and *deadlock* have their obvious meaning. Timed CTL is sufficiently expressive to allow for the formulation of an important set of real-time system properties.

Definition 9.27. Syntax of Timed CTL

Formulae in TCTL are either state or path formulae. TCTL *state formulae* over the set AP of atomic propositions and set C of clocks are formed according to the following grammar:

$$\Phi ::= \text{true} \mid a \mid g \mid \Phi \wedge \Phi \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$