Chapter 4

Timed transition systems

If you put tomfoolery into a computer, nothing comes out of it but tomfoolery. But this tomfoolery, having passed through a very expensive machine, is somehow enobled and no-one dares criticize it. [Pierre Gallois]

Concepts introduced: Finite and infinite systems, reduction using equivalence classes, reachability, regions, zones, DBMs, quotienting, automata.

STATE TRANSITION SYSTEMS are abstract machines that are used in the study of computation. The machine consists of a set of states and transitions between states, and they differ from finite state automata in that state transition systems do not have *accepting* states, and also may have a set of states that is not necessarily finite, or even countable.

4.1 State transition systems

In this chapter, we follow a progressive development, starting from the formal definitions of simple state transition systems, and using these to show how the parallel composition of state transition systems may be used to model a network of state transition systems. Later we continue, developing *timed* transition systems.

The formal definition of a transition system was given in Definition 4 on page 12, and is not repeated here, but please note that S and Act are often *finite* sets, there is often only a single S_{in} state, and the transition relation is often *deterministic* (to be defined soon).

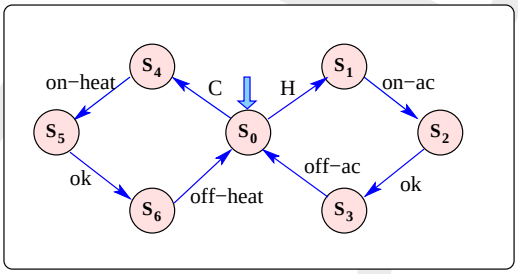


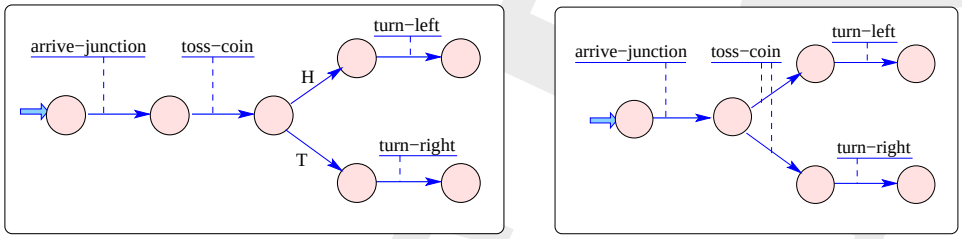
Figure 4.1: Temperature regulator

Consider a control system for a temperature regulator, which controls a heater and an air conditioning unit. In Figure 4.1, we see a transition system that could form the basis for a controller. For clarity, the states have been labelled, and we can use the labels to identify valid and invalid traces:

TRACE: s_4 on-heat s_5 ok s_6 off-heat s_0 ...

NON-TRACE: s_5 off-heat s_6 off-heat s_0 ...

In this system the transition relation is deterministic, i.e. if $s_1 \stackrel{a}{\Longrightarrow} s_2$ and $s_1 \stackrel{a}{\Longrightarrow} s_3$ then $s_2 = s_3$. Non-determinism is useful for getting succinct specifications. When you abstract out elements of a program, this may give rise to non-determinism.



(a) Deterministic TS (b) Non-deterministic TS

Figure 4.2: Abstraction leading to non-determinism

Consider the model in Figure 4.2(a), which deterministically models a system arriving at a road junction, tossing a coin, and then depending on the coin toss (Heads or Tails), turning left or right. The more abstract model in Figure 4.2(b) has less states, and is non-deterministic, and may still be sufficient for modelling purposes.

Definition 18 *A **path** is an allowable sequence of states. Any path starting from an initial state is termed a **run**.*

In a transition system, $\theta = s_0\ s_1\ s_2\ s_3\ \ldots\ s_n$ (written $s_0 \stackrel{*}{\Longrightarrow} s_n$) is a run, with a complete trace of $s_0\ a_1\ s_1\ a_2\ s_2\ a_3\ s_3\ \ldots\ s_{n-1}\ a_n\ s_n$.

Definition 19 *The sequence of actions $a_1\ a_2\ a_3\ \ldots\ a_n$ is termed a **computation**.*

Every run θ induces a computation σ, and given a specific run θ, the corresponding computation σ is not unique. However, if the system is deterministic, for every computation σ, there is a unique run θ.

4.1.1 Parallel composition of transition systems

It is common for transition systems to be presented as an array of smaller transition systems, as we saw in the gate controller system in Section 1.3.2, where three transition systems (Gate, Train, Controller) model the behaviour of the gate, the controller and the train. We now show how to construct the parallel composition of a finite set of such transition systems $\mathrm{ParallelTS} = \mathrm{Gate} \parallel \mathrm{Train} \parallel \mathrm{Controller}$ where $\parallel$ is an operator representing parallel composition. We can construct the parallel composition of finite state transition systems by taking the cartesian product of all the states

of each transition system $\mathcal{S}_{\text{Gate}} \times \mathcal{S}_{\text{Train}} \times \mathcal{S}_{\text{Controller}}$, and then deriving any allowable transitions for each of these states, performing common actions together.

For the ParallelTS system, the cartesian product of all the states gives 72 potential new states (a state space explosion), although only 9 of these are actually used. An efficient way to generate a new transition system is to start from the new starting state ($g_1 t_1 c_1$) synthesized from the starting states (g_1, t_1 and c_1), and then construct all possible future states by taking any actions common to the transition systems. For example, the action available at ($g_1 t_1 c_1$) is `approach`, common to both the Train and the Controller. When we take this action, the next state is ($g_1 t_2 c_2$). This process continues, at each stage constructing any new future state(s), until we have exhausted all possible actions. The end result is seen in Figure 4.3.

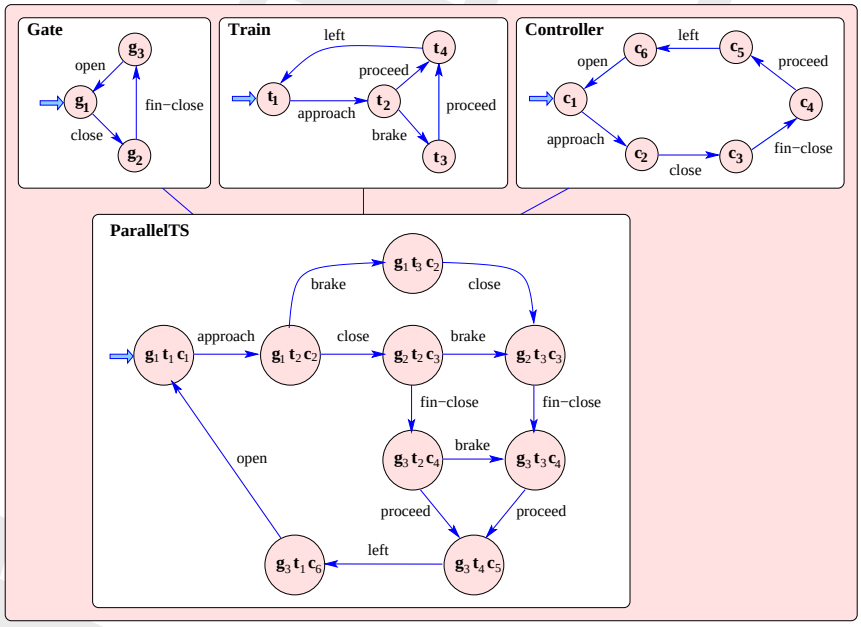


Figure 4.3: Parallel composition of the gate-train-controller

4.2 Timed transition systems

Timed transition systems are transition systems with *clock variables* which are used to record the passage of time. Clock variables operate like hardware timers, can be reset to 0 during a transition, and can be read. Transitions are *guarded* (or constrained) by the current values of the relevant clock variables, which evolve in real-time until reset to 0. To capture all this, transitions are annotated with 3 items: the action, a set of clocks to reset, and a guard predicate over the clock variables:

Definition 20 *A timed transition system* TTS *is a 6-tuple* $(S, S_{\mathrm{in}}, \mathrm{Act}, X, I, \rightarrow)$*:*

1. S, $S_{\mathrm{in}} \subseteq S$ *and* Act *are as defined before*
2. X *is a finite set of* ***clock variables***
3. $I : S \rightarrow \Phi(X)$ *assigns a clock invariant to each state. The clock constraints are limited to constraints of the form*

$$\Phi(X) = x \leq c \mid x \geq c \mid x < c \mid x > c \mid \phi_1 \wedge \phi_2$$

   where $c \in \mathbb{Q}$*.*

4. $\rightarrow$: $S \times \mathrm{Act} \times 2^X \times \Phi(X) \times S$ *is the* ***transition relation****, and* 2^X *is the set of subsets of* X *(the powerset of* X*)*[1]*.*

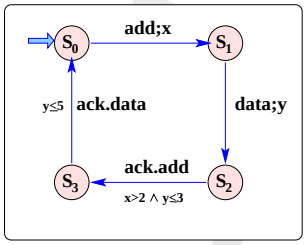
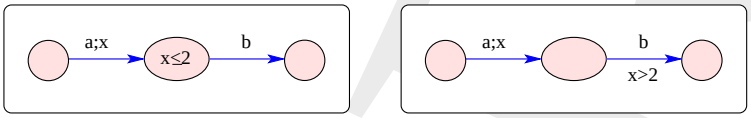


Figure 4.4: Example of TTS

In Figure 4.4 we see a diagram of a simple timed transition system. The actions in this figure are drawn from the set $\mathrm{Act} = \{\mathrm{add}, \mathrm{data}, \mathrm{ack.add}, \mathrm{ack.data}\}$, and the clock variables from $X = \{x, y\}$. When a clock variable is attached to the transition (as in $\mathrm{add}; x$, where the clock variable x is to be associated with this transition), we mean that the clock variable x is to be reset to 0. In this system, $(s_0, \mathrm{add}, \{x\}, \mathrm{True}, s_1)$ and $(s_3, \mathrm{ack.data}, \emptyset, y \leq 5, s_0)$ are valid transitions.

The system can stay in a particular state as long as the state invariant is not violated. For time points which violate the invariant, we expect an output to be enabled, or otherwise we have a *time* deadlock. If more than one output transition is enabled, the choice between the transitions is made non-deterministically.

[1]The notation reflects the size of the powerset.



(a) State invariant (b) Transition guard

Figure 4.5: Guards and state invariants

The state invariants are related to the transition guards. In Figure 4.5(a) we have a state invariant asserting that x should be less than or equal to 2 time units in this state. In Figure 4.5(b) we have a different guard asserting that the transition is enabled if x is more than 2 time units.

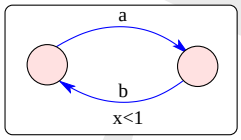


Figure 4.6: A Zeno computation

In Figure 4.6, consider the computation $(b, \frac{1}{2})\ (a, \frac{1}{2})\ (b, \frac{3}{4})\ (a, \frac{3}{4})\ (b, \frac{15}{16}) \ldots$ This could go on forever, and we see that we must model our systems carefully.

4.2.1 Parallel composition of TTS

To compute the parallel composition of timed transition systems, we use the following principles:

1. **Do common actions** together.
2. **Take union** of all the clock variables.
3. **Take conjunction** of all the guards (state invariants).

In Figure 4.7, we show the composition of two timed transition systems into a new timed transition system using this approach.

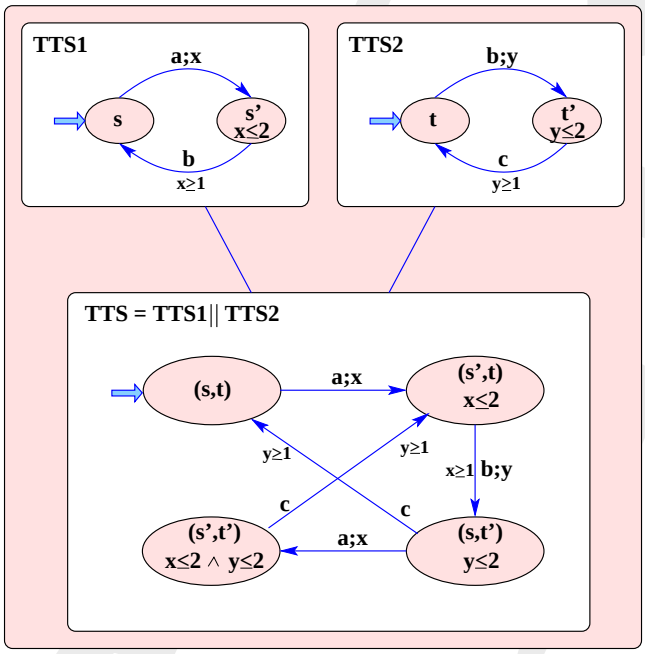


Figure 4.7: Parallel composition of timed transition systems

Given $\mathrm{TTS}_1 = (S_1, S_{0,1}, \mathrm{Act}_1, X_1, I_1, \rightarrow_1)$, $\mathrm{TTS}_2 = (S_2, S_{0,2}, \mathrm{Act}_2, X_2, I_2, \rightarrow_2)$ the product construction $\mathrm{TTS} = \mathrm{TTS}_1 \| \mathrm{TTS}_2 = (S, S_0, \mathrm{Act}, X, I, \rightarrow)$ can be formalized by using the following construction:

- $S = S_1 \times S_2$, $S_0 = S_{0,1} \times S_{0,2}$, $\mathrm{Act} = \mathrm{Act}_1 \cup \mathrm{Act}_2$, $X = X_1 \cup X_2$
- $I(s_1, s_2) = I_1(s_1) \wedge I_2(s_2)$
- Finally, $\rightarrow$ is the least subset of $S \times \mathrm{Act} \times \Phi(X) \times 2^X \times S$ (given $(s_1, a, \phi_1, Y_1, s_1') \in \rightarrow_1$ and $(s_2, b, \phi_2, Y_2, s_2') \in \rightarrow_2$) that satisfies:
  - Case 1: $a = b \in \mathrm{Act}_1 \cap \mathrm{Act}_2$
    - then $((s_1, s_2), a, \phi_1 \wedge \phi_2, Y_1 \cup Y_2, (s_1', s_2')) \in \rightarrow$
  - Case 2: $a \in \mathrm{Act}_1 - \mathrm{Act}_2$
    - then $((s_1, t), a, \phi_1, Y_1, (s_1', t)) \in \rightarrow$ for every $t \in S_2$
  - Case 3: $b \in \mathrm{Act}_2 - \mathrm{Act}_1$
    - then $((t, s_2), b, \phi_2, Y_2, (t, s_2')) \in \rightarrow$ for every $t \in S_1$

4.3 Reduction of timed transition systems

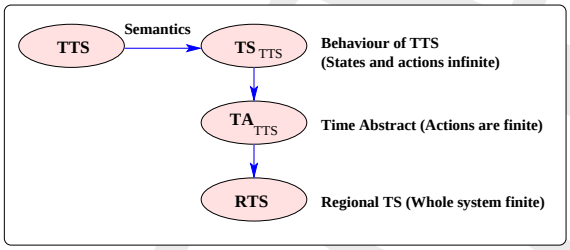


Figure 4.8: Reduction of TTS to a regional automata

The steps we follow to reduce the problem are to show that the TTS can be translated into a finite regional transition system, through intermediate transition systems as seen in Figure 4.8.

4.3.1 From TTS to $\mathrm{TS}_{\mathrm{TTS}}$

The first step is to see how the behaviour of a TTS can be represented by the transition system $\mathrm{TS}_{\mathrm{TTS}}$. The behaviour of timed transition systems is inextricably linked with time, and the values of clock variables at specific times. The transition system $\mathrm{TS}_{\mathrm{TTS}}$ works on (possibly infinite) sets of states $\mathcal{S}$ of the form $S \times V$, where V is a *valuation* (the current values of each clock variable). In Figure 4.4, $(s_1, (2, 5))$ is an example of a state in $\mathcal{S}$.

Given a timed transition system $\mathrm{TTS} = (S, S_{\mathrm{in}}, \mathrm{Act}, X, I, \rightarrow)$, we can derive the associated transition system $\mathrm{TS}_{\mathrm{TTS}} = (\mathcal{S}, \mathcal{S}_0, \mathrm{Act} \cup \mathbb{R}, \Longrightarrow)$ where $\mathcal{S}$ is a (possibly infinite) set of pairs $S \times V$, $\mathcal{S}_0$ is $S_0 \times \{V_0\}$, V are the valuations of the clock variables ($V : X \rightarrow \mathbb{R}$), and finally $\Longrightarrow \subseteq \mathcal{S} \times (\mathrm{Act} \cup \mathbb{R}) \times \mathcal{S}$.

The *timed* behaviour of TTS is defined as the behaviour of $\mathrm{TS}_{\mathrm{TTS}}$ (either as runs or computations, inheriting the same concept from the TS). For example, a possible trace of the timed transition system in Figure 4.4 might be

$$(s_0, (0, 0))\ 1.6\ (s_0, (1.6, 1.6))\ \mathrm{add}\ (s_1, (0, 1.6))\ 2\ (s_1, (2, 3.6))\ \mathrm{data}\ldots$$

We have two types of transitions:

1. **Time passing move:** $(s, V) \overset{\delta}{\Longrightarrow} (s, V + \delta)$, with $\delta \geq 0$
2. **Action move:** $(s, V) \overset{a}{\Longrightarrow} (s', V')$

Two consecutive time passing moves can be amalgamated into one time passing move. For example $(s_0, (0,0))\, 0.6\, (s_0, (0.6, 0.6))\, 0.6\, (s_0, (1.2, 1.2))$ can be amalgamated into $(s_0, (0,0))\, 1.2\, (s_0, (1.2, 1.2))$.

The transition system $\mathrm{TS}_{\mathrm{TTS}} = (\mathcal{S}, \mathcal{S}_0, \mathrm{Act} \cup \mathbb{R}, \Longrightarrow)$ represents the behaviour of a timed transition system $\mathrm{TTS} = (S, S_{\mathrm{in}}, \mathrm{Act}, X, I, \rightarrow)$ in terms of the reachability of states, for both time-passing $(s, V) \stackrel{\delta}{\Longrightarrow} (s, V + \delta)$ and action $(s, V) \stackrel{a}{\Longrightarrow} (s', V')$ transitions, provided there is a transition

$$s \xrightarrow[g]{a;y} s'$$

such that the following conditions are true:

$$V'(x) = \begin{cases} 0 & \text{if } x \in X \\ V(x) & \text{otherwise} \end{cases}$$
$$V \text{ satisfies } g, \text{ the guard for the transition.}$$

In the transition system $\mathrm{TS}_{\mathrm{TTS}}$ we can record runs as for transition systems:

$$(s_0, V_0) \stackrel{\delta_0}{\Longrightarrow} (s_0, V_0') \stackrel{a}{\Longrightarrow} (s_1, V_1) \stackrel{\delta_1}{\Longrightarrow} (s_1, V_1') \stackrel{a_1}{\Longrightarrow} (s_2, V_2)$$

and $s \in S$ is reachable if and only if there is a computation $(s_0, V_0) \stackrel{*}{\Longrightarrow} (s_n, V_n)$ in $\mathrm{TS}_{\mathrm{TTS}}$ such that $s_n = s$.

Definition 21 *$s \in S$ is reachable in a* TTS *if and only if there exists an $(s, V) \in \mathcal{S}$ such that (s, V) is reachable in* $\mathrm{TS}_{\mathrm{TTS}}$.

4.3.2 From $\mathrm{TS}_{\mathrm{TTS}}$ to $\mathrm{TA}_{\mathrm{TTS}}$

We have already seen how the behaviour of TTS can be represented by the transition system $\mathrm{TS}_{\mathrm{TTS}}$, so the next step is to look at the reduction from $\mathrm{TS}_{\mathrm{TTS}}$ to the time-abstract transition system $\mathrm{TA}_{\mathrm{TTS}}$, which has only action moves, and not time-passing moves.

We can derive a time-abstract transition system $\mathrm{TA}_{\mathrm{TTS}} = (\mathcal{S}, \mathcal{S}_0, \mathrm{Act}, \rightsquigarrow)$ from $\mathrm{TS}_{\mathrm{TTS}} = (\mathcal{S}, \mathcal{S}_0, \mathrm{Act} \cup \mathbb{R}, \Longrightarrow)$ where $(s, V) \stackrel{a}{\rightsquigarrow} (s', V')$ if and only if there exists a $\delta \in \mathbb{R}$ such that $(s, V) \stackrel{\delta}{\Longrightarrow} (s, V + \delta) \stackrel{a}{\Longrightarrow} (s', V')$.

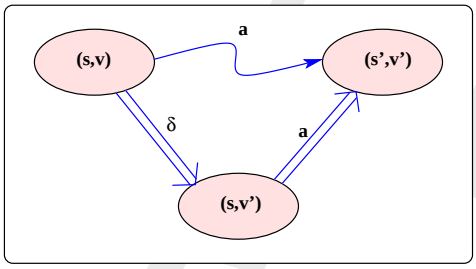


Figure 4.9: The transitions $\rightsquigarrow$ and $\Longrightarrow$

This can be visualized as in Figure 4.9, and the resultant system $\mathrm{TA}_{\mathrm{TTS}}$ has a finite number of actions.

4.3.3 Quotienting

Quotienting, (or partitioning by an equivalence relation[2]) is commonly used to group together objects that are similar in some sense, and hence reduce the complexity of systems. In our domain, we can use quotienting to quotient a big (infinite) transition system into a small (finite) one.

Definition 22 *Given a transition system* $\mathrm{TS} = (S, S_0, \mathrm{Act}, \Longrightarrow)$*, with* $\approx \subseteq S \times S$ *an equivalence relation, then* $\approx$ *is a* ***stable equivalence relation*** *(a bisimulation) if and only if* $s \approx t$ *and* $s \stackrel{a}{\Longrightarrow} s'$ *implies that there exists* t' *such that* $t \stackrel{a}{\Longrightarrow} t'$ *and* $s' \approx t'$ *.*

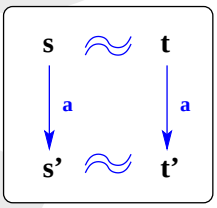


Figure 4.10: Diagram for a stable equivalence relation $\approx$

A category theory diagram shows this construction in Figure 4.10. Since we wish to quotient infinite transition systems into finite ones, we are interested in stable equivalence relations that are finite in some sense.

[2] An equivalence relation on a set X is a binary relation on X that is reflexive, symmetric and transitive.

Definition 23 *Given* $\mathrm{TS} = (S, S_0, \mathrm{Act}, \Longrightarrow)$*, with* $\approx$ *a stable equivalence relation,* $[s]_\approx$ *the equivalence class containing* $s \in S$ *(i.e.* $\{s' \mid s \approx s'\}$*), then* $\approx$ *is a* ***stable equivalence relation of finite index*** *if and only if* $\{[s]_\approx \mid s \in S\}$ *is a finite set.*

So, given a stable equivalence relation of finite index, the process of quotienting (converting a transition system with perhaps an infinite number of states into a finite equivalent one), is relatively easy. Given $\mathrm{TS} = (S, S_0, \mathrm{Act}, \Longrightarrow)$, with $\approx$ a stable equivalence relation of finite index, then a new quotiented transition system is $\mathrm{QTS}_\approx = (QS, QS_0, \mathrm{Act}, \Longrightarrow)$. In this quotiented transition system, $QS = \{[s]_\approx \mid s \in S\}$ and $QS_0 = \{[s_0]_\approx \mid s_0 \in S_0\}$, and we construct $[s]_\approx \stackrel{a}{\Longrightarrow} [s']_\approx$ if and only if there exists $s_1 \in [s]_\approx$ and $s_1' \in [s']_\approx$ such that $s_1 \stackrel{a}{\Longrightarrow} s_1'$ in the transition system TS.

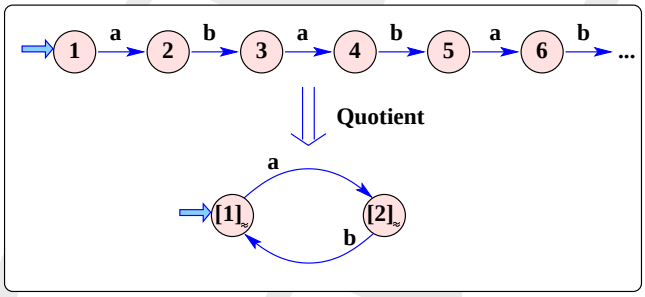


Figure 4.11: Quotienting an infinite transition system

For example, consider the two systems in Figure 4.11. A suitable stable equivalence relation of finite index is *odd* and *even*. More formally, $i \approx j$ if and only if both i and j are odd, or if both i and j are even:

$$\begin{aligned} \{1, 3, 5, \ldots\} &= [1]_\approx \qquad (= [3]_\approx = [5]_\approx = [7]_\approx \ldots) \\ \{2, 4, 6, \ldots\} &= [2]_\approx \qquad (= [4]_\approx = [6]_\approx = [6]_\approx \ldots) \end{aligned}$$

4.3.4 Quotienting and regions

In the previous discussion on quotienting, it is not immediately clear how we can quotient timed transition systems. To understand this we have to return to the original definition of a timed transition system (Definition 20 on page 49). Note the definition of the clock invariant $\Phi(x)$, a set of clock constraints over the set of clocks, is limited to constraints of the form

$$\Phi(X) = x \leq c \mid x \geq c \mid x < c \mid x > c \mid \phi_1 \wedge \phi_2$$

where $c \in \mathbb{Q}$, the set of non-negative rational numbers. There are only a finite number of rationals in any finite timed transition system. We can compute the least common multiple (LCM) k of all the denominators of all the (rational) constants in the original TTS, and then transform our system into a new one TTS' where every term like $x \leq c$ is changed to $x \leq k \cdot c$. We can now assume that all constants are integers, with a new valuation function $V_k(x) = V(x) \cdot k$.

In this new transition system TTS', s is reachable if and only if it was reachable in the original TTS, and we have $(s, V) \approx (s', V')$ if and only if $s = s'$ and $V \equiv_{\mathrm{REG}} V'$ (V is regionally equivalent to V', or V belongs to the same region as V').

For any clock variable x, let C_x be the largest integer appearing in constraints involving x. We now construct a stable equivalence relation of finite index $\equiv_{\mathrm{REG}}$:

Definition 24 *$V \equiv_{\mathrm{REG}} V'$ if and only if the following three conditions are met for all clock variables, x and y:*

1. $\lfloor V(x) \rfloor = \lfloor V'(x) \rfloor$, *or* $V(x) > C_x$ *and* $V'(x) > C_x$.
2. *if* $V(x) \leq C_x$ *and* $V(y) \leq C_y$ *then* $\mathrm{frac}(V(x)) \leq \mathrm{frac}(V(y))$ *if and only if* $\mathrm{frac}(V'(x)) \leq \mathrm{frac}(V'(y))$.
3. *if* $V(x) \leq C_x$, *then* $\mathrm{frac}(V(x)) = 0$ *if and only if* $\mathrm{frac}(V'(x)) = 0$.

For each clock x, we can specify one formula of the form: $c \leq x < c + 1$ where c is in $\{0, 1, \ldots, C_x - 1\}$ or $c = C_x$ or $c > C_x$. For each clock pair we can specify a constraint of the form $x - y = 0$ or $x - y < k$ or $y - x < k$ for a suitable k in case $x \leq C_x$ and $y \leq C_y$.

Finally, given a timed transition system, its (finite) regional transition system can be computed effectively, and hence one can effectively solve verification problems concerning timed transition systems. This is the mathematical basis for the verification tools for timed transition systems and timed automata. As an example, consider a TTS' system of two clocks $\{x, y\}$ with $C_x = 2$ and $C_y = 2$.

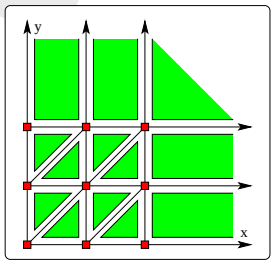


Figure 4.12: Regions for Problem 2

Since we have two variables in the system, the regions can be of dimension 0, 1 or 2, i.e. they can be points, lines or areas.

We can visualize the regions by looking at the diagram in Figure 4.12 where the points are marked with small shaded boxes, the lines are given as lines, and the areas are shaded.

Alternatively, we can enumerate the points, lines, and areas using set enumeration. First the 9 points:

$$\{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)\}$$

There are 22 lines:

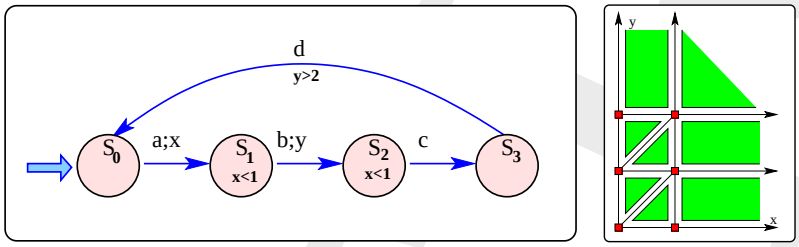
$$\begin{array}{lll}
\{(x,y) \mid y = 0 \wedge 0 < x < 1\} & \{(x,y) \mid y = 0 \wedge 1 < x < 2\} & \{(x,y) \mid y = 0 \wedge x > 2\} \\
\{(x,y) \mid y = 1 \wedge 0 < x < 1\} & \{(x,y) \mid y = 1 \wedge 1 < x < 2\} & \{(x,y) \mid y = 1 \wedge x > 2\} \\
\{(x,y) \mid y = 2 \wedge 0 < x < 1\} & \{(x,y) \mid y = 2 \wedge 1 < x < 2\} & \{(x,y) \mid y = 2 \wedge x > 2\} \\
\{(x,y) \mid x = 0 \wedge 0 < y < 1\} & \{(x,y) \mid x = 0 \wedge 1 < y < 2\} & \{(x,y) \mid x = 0 \wedge y > 2\} \\
\{(x,y) \mid x = 1 \wedge 0 < y < 1\} & \{(x,y) \mid x = 1 \wedge 1 < y < 2\} & \{(x,y) \mid x = 1 \wedge y > 2\} \\
\{(x,y) \mid x = 2 \wedge 0 < y < 1\} & \{(x,y) \mid x = 2 \wedge 1 < y < 2\} & \{(x,y) \mid x = 2 \wedge y > 2\} \\
\{(x,y) \mid 0 < x = y < 1\} & \{(x,y) \mid 1 < x = y < 2\} & \\
\{(x,y) \mid 0 < x = y + 1 < 1\} & \{(x,y) \mid 0 < x + 1 = y < 1\} &
\end{array}$$

And finally the following 13 areas:

$$\begin{array}{ll}
\{(x,y) \mid 0 < x < y < 1\} & \{(x,y) \mid 0 < y < x < 1\} \\
\{(x,y) \mid 1 < x < y < 2\} & \{(x,y) \mid 1 < y < x < 2\} \\
\{(x,y) \mid 0 < x < y - 1 < 1\} & \{(x,y) \mid 0 < y < x - 1 < 1\} \\
\{(x,y) \mid 0 < x - 1 < y < 1\} & \{(x,y) \mid 0 < y - 1 < x < 1\} \\
\{(x,y) \mid y > 2 \wedge 0 < x < 1\} & \{(x,y) \mid y > 2 \wedge 1 < x < 2\} \\
\{(x,y) \mid x > 2 \wedge 0 < y < 1\} & \{(x,y) \mid x > 2 \wedge 1 < y < 2\} \\
\{(x,y) \mid x > 2 \wedge y > 2\} &
\end{array}$$

Another example is the tiny timed transition system shown in Figure 4.13(a).

In this example, $C_x = 1$ and $C_y = 2$, and the corresponding regions look like that shown in Figure 4.13(b). The timed transition system $\mathrm{TTS}_{\mathrm{tiny}}$ may be turned into a



(a) Transition system (b) Regions

Figure 4.13: Tiny timed transition system

regional transition system $\mathrm{RTS}_{\mathrm{tiny}}$ by constructing new states for each of the regions. A regional transition system corresponding to the tiny timed transition system is shown in Figure 4.14

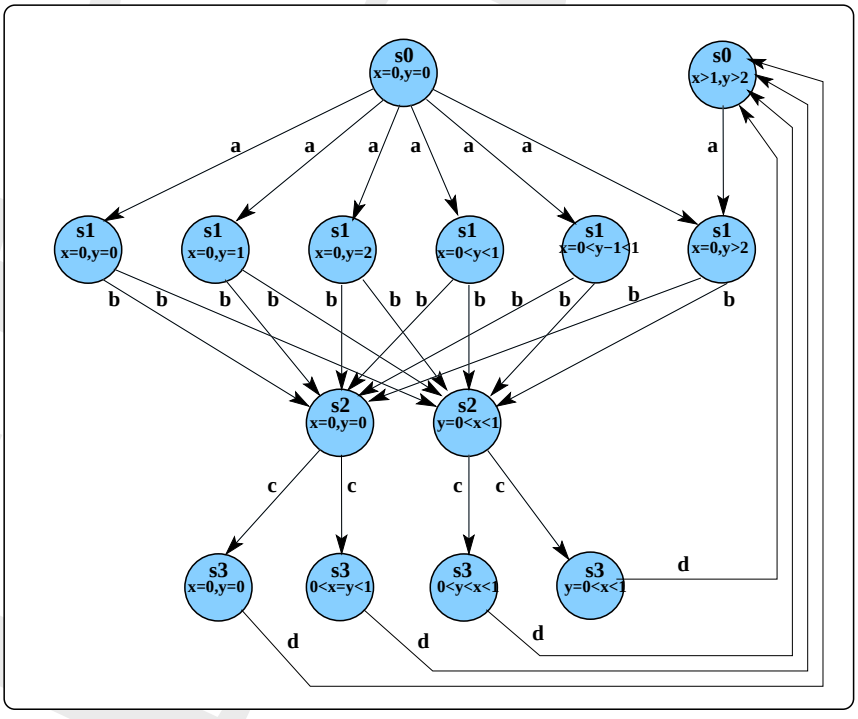


Figure 4.14: Regional transition system

4.3.5 From regions to zones

The region based systems so far are unwieldy to use because the number of regions can be very large. It is exponential in the number of clocks, and in the size of the maximal constraints appearing in the clock constraints. As a result, practical verification of transition systems based on regional transition systems becomes infeasible.

Zones provide a more compact representation, by using equivalence classes of the valuations. The zones can be represented efficiently by DBMs (Difference Bound Matrices), encoding edge-weighted directed graphs. The DBMs admit a canonical representation, and can be manipulated efficiently.

Definition 25 *A zone $\mathcal{Z}$ is a clock constraint of the "two-variable difference" form*

$$\mathcal{Z} ::= x \operatorname{op} c \mid x - y \operatorname{op} c \mid z_1 \wedge z_2$$

where $\operatorname{op} \in \{<, \leq, >, \geq\}$, *and* $c \in \mathbb{N}$.

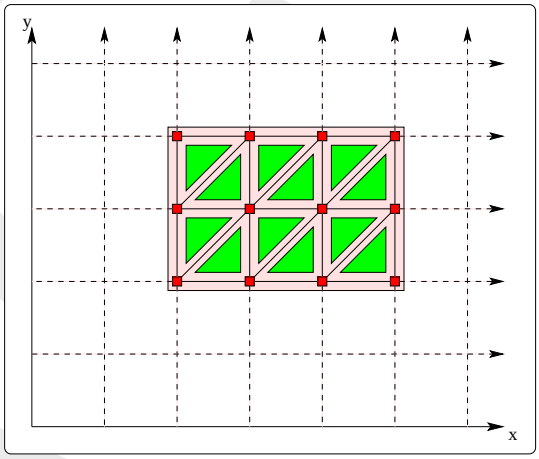


Figure 4.15: Zones and regions

Every region can be encoded as a zone. In Figure 4.15, we see a total of 47 regions (count them) encoded as the single zone

$$(2 \leq x \leq 5) \wedge (2 \leq y \leq 4)$$

It is easy to see that a zone $\mathcal{Z}$ is a convex union (or *hull*) of all the regions $\mathcal{R}$: $\mathcal{Z} = \bigcup_i \mathcal{R}_i$. To encode zones in a DBM, we construct a new clock variable x_0 which will always have the value 0, and then encode all constraints as $x_i - x_j < m$

or $x_i - x_j \leq m$ where $m \in \mathbb{Z}$. For example the following terms on the left are translated to those on the right:

$$\begin{array}{rcl} x_2 & < & 3 \\ x_5 & \geq & 7 \\ x_2 - x_5 & > & 8 \end{array} \quad \Longrightarrow \quad \begin{array}{rcl} x_2 - x_0 & < & 3 \\ x_0 - x_5 & \leq & -7 \\ x_5 - x_2 & < & -8 \end{array}$$

For $n-1$ clock variables, we then write out an $n \times n$ matrix M, with elements drawn from $(\mathbb{Z} \times \{<, \leq\}) \cup \infty$ according to the following rules:

- For constraints like $x_i - x_j < c$, set $M_{i,j} = (c, <)$
- For constraints like $x_i - x_j \leq c$, set $M_{i,j} = (c, \leq)$
- Otherwise set $M_{i,j} = \infty$

If we wished to construct the DBM of the clock zone:

$$(0 \leq x_1 < 1) \wedge (0 < x_2 < 3) \wedge (x_2 - x_1 \geq 1)$$

then the DBM is

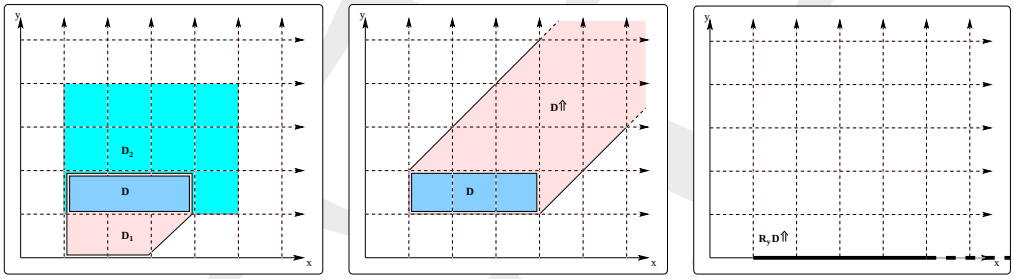
	x_0	x_1	x_2
x_0	$(0, \leq)$	$(0, \leq)$	$(0, <)$
x_1	$(1, <)$	$(0, \leq)$	$(-1, \leq)$
x_2	$(3, <)$	∞	$(0, \leq)$

The canonical DBM for the above zone may be obtained by strengthening all the constraints:

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(0, \leq)$	$(-1, \leq)$
x_1	$(1, <)$	$(0, \leq)$	$(-1, \leq)$
x_2	$(3, <)$	$(3, <)$	$(0, \leq)$

Zones are relatively easily manipulated, and the following three operations are needed for use in evaluating zone transitions:

1. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are two clock zones, then the **intersection** of the zones is a new clock zone $\mathcal{D}_1 \wedge \mathcal{D}_2$.
2. $\mathcal{D} \Uparrow$ is the **time-elapsed zone** defined by $\mathcal{D} \Uparrow = \{V + \delta \mid V \in \mathcal{D}\}$ with $\delta \in \mathbb{R}_{\geq 0}$.
3. The **clock-reset zone** $R_X\mathcal{D}$ is defined by $R_X\mathcal{D} = \{R_X(V) \mid V \in \mathcal{D}\}$ where $R_X(V)(\lambda) = 0$ if $\lambda \in X$ or $R_X(V)(\lambda) = V(\lambda)$ otherwise.



(a) Intersection (b) Time-elapsed (c) Reset $R_y D \Uparrow$

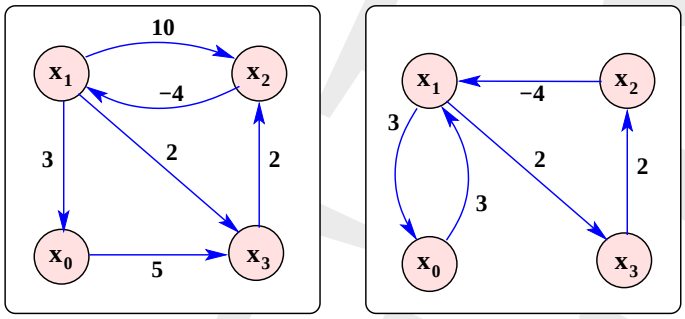
Figure 4.16: Operations on zones

In Figure 4.16 we see progressive operations, starting by calculating zone $\mathcal{D} = \mathcal{D}_1 \wedge \mathcal{D}_2$, and then a corresponding time elapsed zone $\mathcal{D} \Uparrow$ extending out in time at an angle of 45^o, before finally doing the reset operation for the y variable.

The zones can also be optimized, and it is useful if we use a representation of zones that is canonical. We do not want two different zones to represent the same set of valuations (i.e. $(y - x \leq 3, x = 2, y = 4)$ the same as $(y - x = 2, x = 2, y = 4)$.

Definition 26 *A zone is closed if no constraint can be strengthened without reducing the set of associated valuations.*

Two closed zones are equivalent if and only if they are identical. A graph representation of a zone can be used to find the closed zone version of a zone.



(a) Non optimal zone (b) Reduced zone

Figure 4.17: Graph representation of zones

In Figure 4.17(a), we see a graph representation for a zone, in which the nodes represent the clock variables, and the arcs represent the differences. For example the arc $x_1 \xrightarrow{10} x_2$ corresponds to the constraint $x_2 - x_1 < 10$. A shortest path reduction is performed on the graph, giving Figure 4.17(b), where redundant edges are removed when they can be. For example $x_1 \xrightarrow{10} x_2$ is replaced by $x_1 \xrightarrow{2} x_3 \xrightarrow{2} x_2$.

4.3.6 Reachability, safety and automata

An automata is a state transition system with some set of accepting states, which may be used to distinguish between *good* and *bad* computations. We can use automata matching a particular transition system to specify *desired* behaviour of the system.

Definition 27 *A finite automaton is a 5-tuple* $(Q, \Sigma, \Delta, q_0, F)$*, where*

1. Q *is a finite set called the* ***states***
2. Σ *is a finite set called the* ***alphabet***
3. $\Delta : Q \times \Sigma \to Q$ *is the* ***transition function***
4. q_0 *is the* ***start state***
5. $F \subseteq Q$ *is the set of* ***accepting states***

In [Lam77], the ideas of safety and liveness were introduced, identifying two types of behaviours that require different analysis methods. A safety property is like “something bad doesn’t happen”, whereas liveness is like “something bad (or good) must

eventually happen". We can often formulate safety properties in terms of the reachability of a state.

We can use automata to formalize these questions, in a form like “Is there a run of the automaton that leads to the (desired) accepting state?”, or “Is there a run of the automaton that leads to an accepting state in which property P holds?”. These are examples of the reachability problem.

It is easy to see from this that reachability problems have clear links to automata theory, and we could easily cast a lot of this discussion in terms of the languages accepted by (finite) automata. However to reason about liveness properties, we need to consider infinite sequences.

A Büchi automaton is an extension of a finite state automaton to one which accepts an infinite input sequence if, and only if, there is a run of the automaton which has infinitely many states in the set of final states.

Definition 28 *A Büchi automaton is a 5-tuple $(Q, \Sigma, \Delta, q_0, F)$, as for a regular automaton, but with F interpreted differently. In particular $s_0\, a_0\, s_1\, a_1\, s_2\, \ldots$ is an accepting infinite trace if*

1. $s_0 \in Q$
2. $(s_i, a_i, s_{i+1}) \in \Delta$ *for all* i
3. *For infinitely many j, the state s_j is in F*

They are useful for specifying behavior of nonterminating systems, such as hardware (electronic circuits) or operating systems. For example, you may want to specify a property like “for every measurement, a recording eventually follows”, or the reverse “there is a measurement which is not followed by a recording". For the second example, an argument limited to finite sequences cannot satisfy this property.

Later, in Chapter 5, we will consider the analysis of liveness properties.

4.4 Summary of topics

In this section, we introduced the following topics:

State transition systems. *Basic properties of state transition systems, Parallel composition, runs and computations.*
Timed transition systems. *Basic properties of timed transition systems, Parallel composition.*
Reduction of timed transition systems. *Reducing infinite systems to finite (regional) transition systems, regions and zones.*