1
Introduction

1.1 What is a real-time system?

This book is about the design of certain kinds of reactive systems. A *reactive system* interacts with its environment by reacting to inputs from the environment with certain outputs. Usually, a reactive system is not supposed to stop but should be continuously ready for such interactions. In the real world there are plenty of reactive systems around. A vending machine for drinks should be continuously ready for interacting with its customers. When a customer inputs suitable coins and selects "coffee" the vending machine should output a cup of hot coffee. A traffic light should continuously be ready to react when a pedestrian pushes the button indicating the wish to cross the street. A cash machine of a bank should continuously be ready to react to customers' desire for extracting money from their bank account.

Reactive systems are seen in contrast to *transformational systems*, which are supposed to compute a single input–output transformation that satisfies a certain relation and then terminate. For example, such a system could input two matrices and compute its product.

We wish to design reactive systems that interact in a well-defined relation to the real, physical time. A *real-time system* is a reactive system which, for certain inputs, has to compute the corresponding outputs within given time bounds. An example of a real-time system is an *airbag*. When a car is forced into an emergency braking its airbag has to unfold within 300 milliseconds to protect the passenger's head. Thus there is a tight upper time bound for the reaction. However, there is also a lower time bound of 100 milliseconds. If the airbag unfolds too early, it will deflate and thus lose its protective impact before the passenger's head sinks into it. This shows that both *lower* and *upper* time bounds are important. The outputs of a real-time system may depend on the *behaviour* of its inputs over time. For instance, a *watchdog*

has to raise an alarm (output) if an input signal is absent for a period of t seconds.

Real-time constraints often arise indirectly out of safety requirements. For example, a gas burner should avoid a critical concentration of unburned gas in the air because this could lead to an explosion. This is an untimed safety requirement. To achieve it, a controller for a gas burner could react to a flame failure by shutting down the gas valve for a *sufficiently large period of time* so that the gas can evaporate during that period. This way the safety requirement is reduced to a real-time constraint.

The gas burner is an example of a *safety critical* system: a malfunction of such a system can cause loss of goods, money, or even life. Other examples are the airbag in a car, traffic controllers, auto pilots, and patient monitors.

Real-time constraints are sometimes classified into *hard* and *soft*. Hard constraints must be fulfilled without exception, whereas soft ones should not be violated. For example, a car control system *should* meet the real-time requirements for the air condition, but *must* meet the real-time constraints for the airbag.

In constructing a real-time system the aim is to control a physically existing environment, the *plant*, in such a way that the controlled plant satisfies all desired timing requirements: see Figure 1.1.

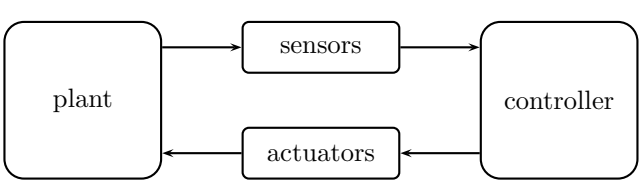

Fig. 1.1. Real-time system

The *controller* is a digital computer that interacts with the plant through *sensors* and *actuators*. By reading the sensor values the controller inputs information about the current state of the plant. Based on this input the controller can manipulate the state of the plant via the actuators. A precise model of controller, sensors, and actuators has to take *reaction times* of these components into account because they cannot work arbitrarily fast.

In many cases the plant is distributed over different physical locations. Also the controller might be implemented on more than one machine. Then one talks of *distributed systems*. For instance, a railway station consists of many points and signals in the field together with several track sensors and actuators. Often the controller is hidden to human beings. Such real-time

systems are called *embedded systems*. Examples of embedded systems range from controllers in washing machines to airbags in cars.

When we model the plant in Figure 1.1 in more detail we arrive at *hybrid systems*. These are defined as reactive systems consisting of continuous and discrete components. The continuous components are time-dependent physical variables of the plant ranging over a continuous value set, like temperature, pressure, position, or speed. The discrete component is the digital controller that should influence the physical variables in a desired way. For example, a heating system should keep the room temperature within certain bounds. Real-time systems are systems with at least one continuous variable, that is time. Often real-time systems are obtained as abstractions from the more detailed hybrid systems. For example, the exact position of a train relative to a railroad crossing may be abstracted into the values *far_away, near_by*, and *crossing*.

Figure 1.2 summarises the main classes of systems discussed above and shows their containment relations: hybrid systems are a special class of real-time systems, which in turn are a special class of reactive systems.

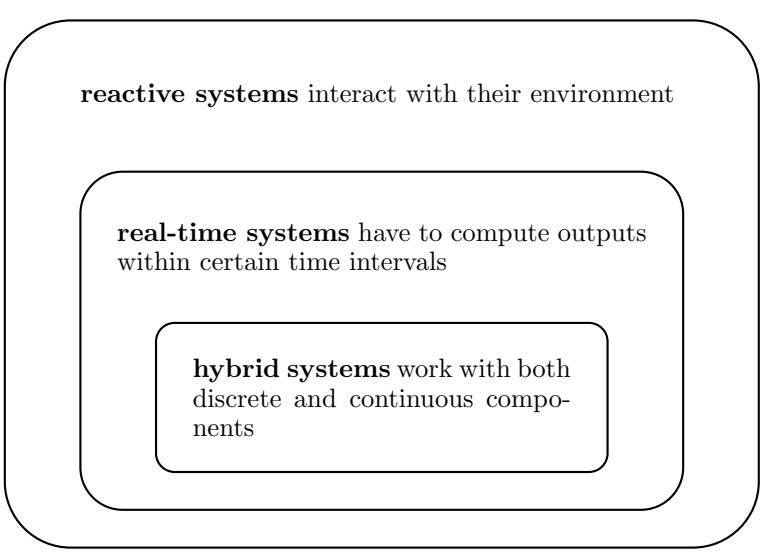

Fig. 1.2. Classes of systems

Since real-time systems often appear in safety-critical applications, their design requires a high degree of precision. Here, formal methods based on mathematical models of the system under design are helpful. They allow the designer to specify the system at different levels of abstraction and to formally verify the consistency of these specifications before implementing

them. In recent years significant advances have been made in the maturity of formal methods that can be applied to real-time systems.

When considering formal methods for specifying and verifying systems we have the reverse set of inclusions of Figure 1.2, as shown in Figure 1.3: formal methods for hybrid systems can also be used to analyse real-time systems, and formal methods for real-time systems can also be used to analyse reactive systems.

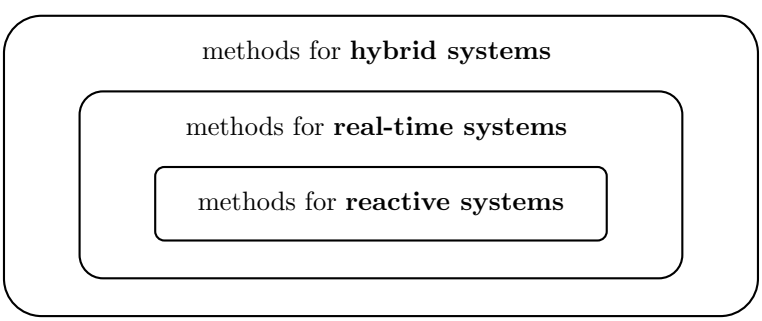

Fig. 1.3. Formal methods for systems classes

1.2 System properties

To describe real-time systems formally, we start by representing them by a collection of time-dependent *state variables* or *observables* obs, which are functions

$$\mathsf{obs} : \mathsf{Time} \longrightarrow \mathcal{D}$$

where Time denotes the time domain and $\mathcal{D}$ is the data type of obs. Such observables describe an infinite system behaviour, where the current data values are recorded at each moment of time.

For example, a gas valve might be described using a Boolean, i.e. {0,1}-valued observable

$$G : \mathsf{Time} \longrightarrow \{0, 1\}$$

indicating whether gas is present or not, a railway track by an observable

$$\mathsf{Track} : \mathsf{Time} \longrightarrow \{empty, appr, cross\}$$

where *appr* means a train is approaching and *cross* means that it is crossing the gate, and the current communication trace of a reactive system by an observable

$$\mathsf{trace} : \mathsf{Time} \longrightarrow Comm^*$$

where $Comm^*$ denotes the set of all finite sequences over a set $Comm$ of possible communications. Thus depending on the choice of observables we can describe a real-time system at various levels of detail.

There are two main choices for time domain Time:

- **discrete time:** $\mathsf{Time} = \mathbb{N}$, the set of natural numbers, and
- **continuous time:** $\mathsf{Time} = \mathbb{R}_{\geq 0}$, the set of non-negative real numbers.

A discrete-time model is appropriate for specifications which are close to the level of implementation, where the time rate is already fixed. For higher levels of specifications continuous time is well suited since the plant models usually use continuous-state variables. Moreover, continuous-time models avoid a too-early introduction of hardware considerations. Throughout this book we shall use the continuous-time model and consider discrete time as a special case.

To describe desirable properties of a real-time system, we constrain the values of their observables over time, using formulas of a suitable logic. In this introduction we simply take *predicate logic* involving the usual logical connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\Longrightarrow$ (implication), and $\Longleftrightarrow$ (equivalence) as well as the quantifiers $\forall$ (for all) and $\exists$ (there exists). When expressing properties of real-time systems quantification will typically range over time points, i.e. elements of the time domain Time. Later in this book we introduce dedicated notations for specifying real-time systems.

In the following we discuss some typical types of properties. For reactive systems properties are often classified into safety and liveness properties. For real-time systems these concepts can be refined.

Safety properties. Following L. Lamport, a safety property states that *something bad must never happen*. The "bad thing" represents a critical system state that should never occur, for instance a train being inside a crossing with the gates open. Taking a Boolean observable $C : \mathsf{Time} \longrightarrow \{0, 1\}$, where $C(t) = 1$ expresses that at time t the system is in the critical state, this safety property can be expressed by the formula

$$\forall t \in \mathsf{Time} \bullet \neg C(t). \tag{1.1}$$

Here $C(t)$ abbreviates $C(t) = 1$ and thus $\neg C(t)$ denotes that at time t the system is not in the critical state. Thus for all time points it is not the case that the system is in the critical state.

In general, a safety property is characterised as a property that

can be *falsified* in bounded time. In case of (1.1) exhibiting a single time point t_0 with $C(t_0)$ suffices to show that (1.1) does not hold.

In the example, a crossing with permanently closed gates is safe, but it is unacceptable for the waiting cars and pedestrians. Therefore we need other types of properties.

Liveness properties. Safety properties state what may or may not occur, but do not require that anything ever does happen. Liveness properties state what must occur. The simplest form of a liveness property guarantees that *something good eventually does happen.* The "good thing" represents a desirable system state, for instance the gates being open for the road traffic. Taking a Boolean observable $G : \mathsf{Time} \longrightarrow \{0, 1\}$, where $G(t) = 1$ expresses that at time t the system is in the good state, this liveness property can be expressed by the formula

$$\exists t \in \mathsf{Time} \bullet G(t). \tag{1.2}$$

In other words, there exists a time point in which the system is in the good state. Note that this property cannot be falsified in bounded time. If for any time point t_0 only $\neg G(t)$ has been observed for $t \leq t_0$, we cannot complain that (1.2) is violated because *eventually* does not say how long it will take for the good state to occur.

Such liveness property is not strong enough in the context of real-time systems. Here one would like to see a time bound when the good state occurs. This brings us to the next kind of property.

Bounded response properties. A bounded response property states that a desired system reaction to an input occurs *within a time interval* $[b, e]$ with lower bound $b \in \mathsf{Time}$ and upper bound $e \in \mathsf{Time}$ where $b \leq e$. For example, whenever a pedestrian at a traffic light pushes the button to cross the road, the light for pedestrians should turn *green* within a time interval of, say, $[10, 15]$. The need for an upper bound is clear: the pedestrian wants to cross the road within a short time (and not *eventually*). However, also a lower bound is needed because the traffic light must not change from *green* to *red* instantaneously, but only after a *yellow* phase of, say, 10 seconds to allow cars to slow down gently.

With $P(t)$ representing the pushing of the button at time t and $G(t)$ representing a green traffic light for the pedestrians at time t, we can express the desired property by the formula

$$\forall t_1 \in \mathsf{Time} \bullet (P(t_1) \Longrightarrow \exists t_2 \in [t_1 + 10, t_1 + 15] \bullet G(t_2)). \tag{1.3}$$

Note that this property can be falsified in bounded time. When for some time point t_1 with $P(t_1)$ we find out that during the time interval $[t_1 + 10, t_1 + 15]$ no green light for the pedestrians appeared, property (1.3) is violated.

Duration properties. A duration property is more subtle. It requires that for observation intervals $[b, e]$ satisfying a certain condition $A(b, e)$ the *accumulated time* in which the system is in a certain critical state has an upper bound $u(b, e)$. For example, the leak state of a gas burner, where gas escapes without a flame burning, should occur at most 5% of the time of a whole day.

To measure the accumulated time t of a critical state $C(t)$ in a given interval $[b, e]$ we use the integral notion of mathematical calculus:

$$\int_b^e C(t)dt.$$

Then the duration property can be expressed by a formula

$$\forall b, e \in \mathsf{Time} \bullet \left(A(b, e) \Longrightarrow \int_b^e C(t)dt \leq u(b, e) \right). \tag{1.4}$$

Again this property can be falsified in finite time. If we can point out an interval $[b, e]$ satisfying the condition $A(b, e)$ where the value of the integral is too high, property (1.4) is violated.

1.3 Generalised railroad crossing

This case study is due to C. Heitmeyer and N. Lynch [HL94]. It concerns a railroad crossing with a physical layout as shown in Figure 1.4, for the case of two tracks. In the safety-critical area "Cross" the road and the tracks intersect. The gates (indicated by "Gate") can move from fully "closed" (where the angle is 0°) to fully "open" (where the angle is 90°). Moving the gates up and down takes time. Sensors at the tracks will detect whether a train is approaching the crossing, i.e. entering the area marked by "Approach".

1.3.1 The problem

Given are two time parameters $\xi_1, \xi_2 > 0$ describing the reaction times needed to open and close the gates, respectively. In the following problem description time intervals are used that collect all time points in which at least one train is in the area "Cross". These are called *occupancy intervals* and denoted by $[\tau_i, \nu_i]$ where the subscripts $i \in \mathbb{N}$ enumerate their successive

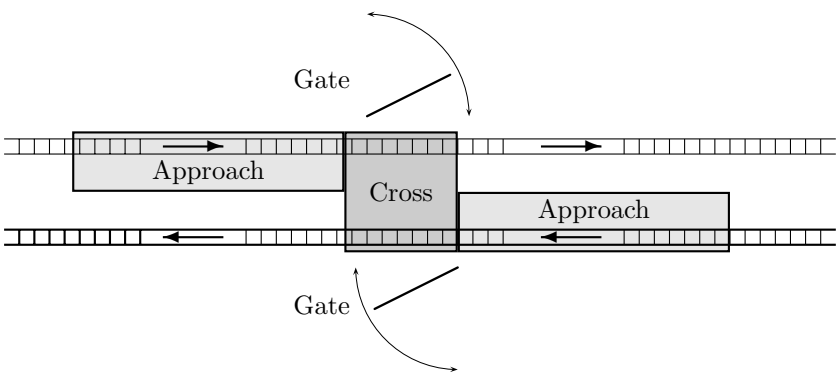

Fig. 1.4. Generalised railroad crossing

occurrences. As usual, a closed interval $[\tau_i, \nu_i]$ is the set of all time points t with $\tau_i \leq t \leq \nu_i$. Moreover, for a time point t let $g(t)$ denote the angle of the gates, ranging from 0 (closed) to 90 (open).

The task is to construct a controller that operates the gates of the railroad crossing such that the following two properties hold for all time points t:

- **Safety:** $t \in \bigcup_{i\in\mathbb{N}} [\tau_i, \nu_i] \Longrightarrow g(t) = 0$, i.e. the gates are closed inside all occupancy intervals.
- **Utility:** $t \notin \bigcup_{i\in\mathbb{N}} [\tau_i - \xi_1, \nu_i + \xi_2] \Longrightarrow g(t) = 90$, i.e. outside the occupancy intervals extended by the reaction times ξ_1 and ξ_2 the gates are open.

This problem statement is taken from the article of Heitmeyer and Lynch [HL94]. Note that the safety and utility properties are consistent, i.e. the gate is never required to be simultaneously open and closed. To see this, take a time point t satisfying the precondition (the left-hand side of the implication) of the utility property. Then in particular,

$$t \notin \bigcup_{i\in\mathbb{N}} [\tau_i, \nu_i],$$

which implies that t does not satisfy the precondition of the safety property. Thus never both $g(t) = 0$ and $g(t) = 90$ are required.

Note, however, that depending on the choice of the time parameters ξ_1, ξ_2 and the timing of the trains it may well be that in between two successive trains there is not enough time to open the gate, i.e. two successive time intervals

$$[\tau_i - \xi_1, \nu_i + \xi_2] \quad \text{and} \quad [\tau_{i+1} - \xi_1, \nu_{i+1} + \xi_2]$$

may overlap (see also Figure 1.5).

In the following we formalise and analyse this case study in terms of predicate logic over suitable observables.

1.3.2 Formalisation

The railroad crossing can be described by two observables:

$$\begin{array}{rcll} \mathsf{Track} : \mathsf{Time} & \longrightarrow & \{\mathsf{empty}, \mathsf{appr}, \mathsf{cross}\} & \text{(state of the track)} \\ g : \mathsf{Time} & \longrightarrow & [0, 90] & \text{(angle of the gate).} \end{array}$$

Note that via the three values of the observable Track we have abstracted from further details of the plant like the exact position of the train on the track. The value empty expresses that no train is in the areas "Approach" or "Cross", the value appr expresses that a train is in the area "Approach" and none is in "Cross", and the value cross expresses that a train is in the area "Cross". The observable g ranges over all values of the gate angle in the interval $[0, 90]$. We will use the following abbreviations:

$$\begin{array}{rcl} E(t) & \text{stands for} & \mathsf{Track}(t) = \mathsf{empty} \\ A(t) & \text{stands for} & \mathsf{Track}(t) = \mathsf{appr} \\ Cr(t) & \text{stands for} & \mathsf{Track}(t) = \mathsf{cross} \\ O(t) & \text{stands for} & g(t) = 90 \\ Cl(t) & \text{stands for} & g(t) = 0. \end{array}$$

Requirements. With these observables and abbreviations we can specify the requirements of the generalised railroad crossing in predicate logic. The safety requirement is easy to specify:

$$\mathsf{Safety} \stackrel{\text{def}}{\Longleftrightarrow} \forall t \in \mathsf{Time} \bullet Cr(t) \Longrightarrow Cl(t) \tag{1.5}$$

where $\stackrel{\text{def}}{\Longleftrightarrow}$ means *equivalence by definition*. Thus whenever a train is in the crossing the gates are closed. Note that this formula is logically equivalent to the property **Safety** above because by the definition of $Cr(t)$ we have

$$\forall t \in \mathsf{Time} \bullet Cr(t) \iff t \in \bigcup_{i \in \mathbb{N}} [\tau_i, \nu_i],$$

i.e. $Cr(t)$ holds if and only if t is in one of the occupancy intervals.

Without the reaction times ξ_1 and ξ_2 of the gate the utility requirement could simply be specified as

$$\forall t \in \mathsf{Time} \bullet \neg Cr(t) \Longrightarrow O(t).$$

However, the property **Utility** refers to (the complements of) the intervals $[\tau_i - \xi_1, \nu_i + \xi_2]$, which are not directly expressible by a certain value of the observable Track. In Figure 1.5 the occupancy intervals $[\tau_i, \nu_i]$ and their extensions to $[\tau_i - \xi_1, \nu_i + \xi_2]$ are shown for $i = 0, 1, 2$. Only outside of the latter intervals, in the areas exhibited by the thick line segments, are the gates required to be open.

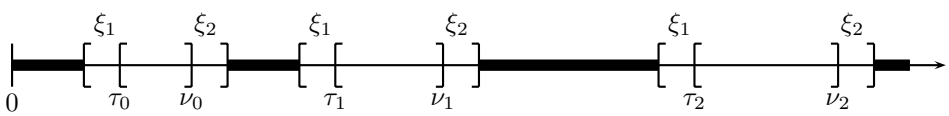

Fig. 1.5. Utility requirement

We specify this as follows. Consider a time point t. If in a suitable time interval containing t there is no train in the crossing then $O(t)$ should hold. Calculations show that this interval is given by $[t - \xi_2, t + \xi_1]$. Thus $\neg Cr(\tilde{t})$ should hold for all time points $\tilde{t}$ with $t - \xi_2 \leq \tilde{t} \leq t + \xi_1$. This is expressed by the following formula:

$$\begin{aligned} \mathsf{Utility} \overset{\text{def}}{\iff} \; & \forall t \in \mathsf{Time} \bullet \\ & (\forall \tilde{t} \in \mathsf{Time} \bullet t - \xi_2 \leq \tilde{t} \leq t + \xi_1 \implies \neg Cr(\tilde{t})) \\ & \implies O(t). \end{aligned} \tag{1.6}$$

Note the subtlety that $t - \xi_2$ may be negative whereas $\tilde{t} \in \mathsf{Time}$ is by definition non-negative. It can be shown that this formula Utility is equivalent to the property **Utility** above (see Exercise 1.2).

For the generalised railroad crossing all functions Track and g are admissible that satisfy the two requirements above. These functions can be seen as *interpretations* of the observables Track and g. They are presented as *timing diagrams*. Figure 1.6 shows an admissible interpretation of Track and g.

Assumptions. In this case study Track is an *input observable* which can be read but not influenced by the controller. By contrast, g is an *output observable* since it can be influenced by the controller via actuators. The correct behaviour of the controller often depends on some assumptions about the input observables. Here we make the following assumptions about Track:

- Initially the track is empty: $\mathsf{Init} \overset{\text{def}}{\iff} E(0)$.