Real-Time Systems

Formal Specification and Automatic Verification

E.-R. Olderog and H. Dierks

CAMBRIDGE

CAMBRIDGE www.cambridge.org/9780521883337

Real-Time Systems

Real-time systems need to react to certain input stimuli within given time bounds. For example, an airbag in a car has to unfold within 300 milliseconds in a crash. There are many embedded safety-critical applications and each requires real-time specification techniques. This textbook introduces three of these techniques, based on logic and automata: Duration Calculus, Timed Automata, and PLC-Automata.

The techniques are brought together to form a seamless design flow, from real-time requirements specified in the Duration Calculus, via designs specified by PLC-Automata, and into source code for hardware platforms of embedded systems. The syntax, semantics, and proof methods of the specification techniques are introduced; their most important properties are established; and real-life examples illustrate their use. Detailed case studies and exercises conclude each chapter.

Ideal for students of real-time systems or embedded systems, this text will also be of great interest to researchers and professionals in transportation and automation.

E.-R. OLDEROG is Professor of Computer Science at the University of Oldenburg, Germany. In 1994 he was awarded the Leibniz Prize of the German Research Council (DFG).

H. DIERKS is a researcher currently working with OFFIS, a technology transfer institute for computer science in Oldenburg, Germany.

REAL-TIME SYSTEMS

Formal Specification and Automatic Verification

ERNST-RÜDIGER OLDEROG[1] AND HENNING DIERKS[2]

[1] Department of Computing Science, University of Oldenburg, Germany

[2] OFFIS, Oldenburg, Germany

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK
Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521883337

First published in print format 2008

ISBN-13 978-0-511-42693-3 eBook (Adobe Reader)

ISBN-13 978-0-521-88333-7 hardback

Contents

Preface

Computers are used more and more to provide high-quality and reliable products and services, and to control and optimise production processes. Such computers are often embedded into the products and thus hidden to the human user. Examples are computer-controlled washing machines or gas burners, electronic control units in cars needed for operating airbags and braking systems, signalling systems for high-speed trains, or robots and automatic transport vehicles in industrial production lines.

In these systems the computer continuously interacts with a physical environment or plant. Such systems are thus called reactive systems. Moreover, common to all these applications is that the computer reactions should obey certain timing constraints. For example, an airbag has to unfold within milliseconds, not too early and not too late. Reactive systems with such constraints are called real-time systems. They often appear in safety-critical applications where a malfunction of the controller will cause damage and risk the lives of people. This is immediately clear for all applications in the transport sector where computers control cars, trains and planes.

Therefore the design of real-time systems 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.

Structure of this book

In this advanced textbook we shall present three such formal approaches:

- Duration Calculus (DC for short), a logic and calculus for specifying high-level requirements of real-time systems;
- timed automata (TA for short), a state-transition model of real-time systems with the advantage of elaborate tool support for the automatic verification of real-time properties;
- PLC-Automata, a state-transition model of real-time systems with the advantage of being implementable, for example in the programming language C or on Programmable Logic Controllers (PLCs for short), a hardware platform that is widespread in the automation industry.

This book is the first one that presents the above three approaches to the specification of real-time systems in a coherent way. This is achieved by combining the approaches into a design method for real-time systems, reaching from requirements down to executable code as illustrated in Figure 0.1. Here:

- Real-time requirements are specified in the Duration Calculus or subsets thereof.
- Designs are specified by PLC-Automata.
- Implementations are written as C programs with timers or as programs that are executable on PLCs.
- Automatic verification of requirements is performed using the model-checking tool UPPAAL for timed automata.
- A tool MOBY/RT, built for PLC-Automata, allows the user to invoke algorithms for generating C or PLC code from such automata, and to automatically verify properties specified in a subset of Duration Calculus by using UPPAAL as a back-end verification engine.

The connection is that PLC-Automata have both a semantics in terms of the Duration Calculus and an equivalent one in terms of timed automata. To verify that a PLC-Automaton satisfies a given real-time requirement expressed in the Duration Calculus, there are two possibilities: either a proof can be conducted in the Duration Calculus exploiting the corresponding semantics of the PLC-Automaton, or, for certain types of requirement, an automatic verification is possible using the tool UPPAAL and the timed automata semantics of the PLC-Automaton.

How to read this book

The titles and dependencies of the chapters are shown in Figure 0.2. First, the introduction in Chapter 1 should be read. Here two case studies (railroad

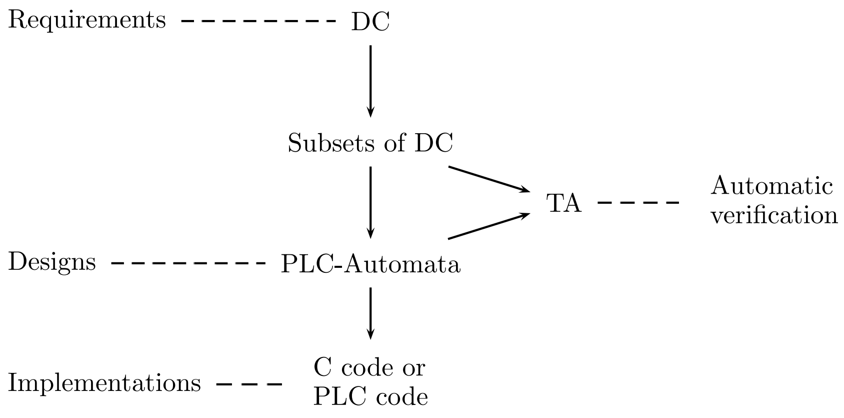

Fig. 0.1. Overview of design method

crossing and gas burner) provide a feeling for the delicacies of real-time systems. Then one can continue with Chapter 2 (Duration Calculus) or Chapter 4 (Timed automata).

Chapter 2 presents the basic knowledge of the Duration Calculus. First, the syntax and semantics of the logic are defined. Then the proof rules of the calculus are introduced, including a simple induction rule. These rules are applied to the case study of the gas burner.

Chapter 3 presents advanced topics on the Duration Calculus. First, decidability results are discussed for the cases of discrete and continuous time domains. Then a subset of the Duration Calculus that is closer to the implementation level is presented, the so-called DC implementables. Finally, Constraint Diagrams are introduced as a graphic representation for requirements with a semantics in the Duration Calculus.

Chapter 4 presents the basic facts of timed automata. In particular, the most prominent result of timed automata is shown: the decidability of the reachability problem. It is then explained which variant of timed automata and properties the model checker UPPAAL can decide.

Chapter 5 introduces PLC-Automata as a class of implementable real-time automata. First, these automata are motivated using an example of a real-time filter. Then it is described how PLC-Automata can be compiled into code that is executable on Programmable Logic Controllers (PLCs). To link the PLC-Automata with the Duration Calculus, their semantics are defined in terms of this logic. As a consequence, a general result estimating the reaction times of PLC-Automata to input stimuli can be proved. Also, an algorithm is discussed that synthesises a PLC-Automaton from a given set of DC implementables provided this set is consistent. Finally, hierarchical PLC-Automata are defined.

Chapter 6 ties together the results of Chapters 4 and 5 for the purposes of automatic verification. It turns out that certain real-time properties of PLC-Automata can be proven automatically using the model checker UPPAAL for timed automata. To this end, an alternative and equivalent semantics of PLC-Automata in terms of timed automata is defined. Then it is shown that real-time requirements expressed in a subset of Constraint Diagrams can be verified against PLC-Automata by checking the reachability of certain states with UPPAAL. This is all supported by the tool MOBY/RT, which is described briefly as well. Also, MOBY/RT enables the user to compile PLC-Automata into PLC code or C code.

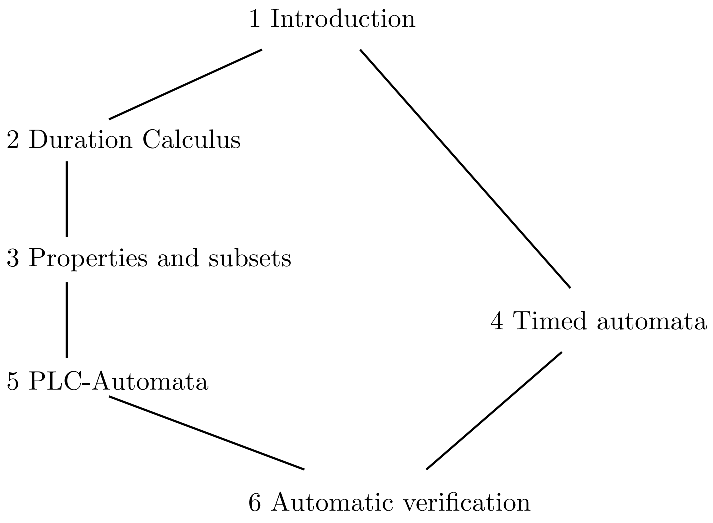

Fig. 0.2. Dependency of chapters

Actually, only Section 5.5 (Synthesis) of Chapter 5 depends on Section 3.2 (DC implementables) of Chapter 3. The remainder of Chapter 5 can thus also be read immediately after Chapter 2.

Intended audience

This textbook is appropriate for either a course on formal methods for real-time systems in the upper division of undergraduate studies or for graduate studies in computer science and engineering. It can also be used for self study, and will be of interest for engineers of embedded real-time systems. Readers are expected to have a basic understanding of mathematical and logical notations.

Courses based on this book

Our own course on real-time systems at the University of Oldenburg is for M.Sc. and advanced B.Sc. students in computer science with an interest in embedded systems; it proceeds as follows:

Course at Oldenburg	
Introduction	1
Duration Calculus	2
Properties and subsets	3.1–3.2
Timed automata	4
PLC-Automata	5.1–5.5
Automatic verification	6 (only short indication)

The course takes one semester with three hours of lectures and one hour of exercises per week.

At Oldenburg an in-depth study of Chapter 6 (Automatic verification) with the use of the tools UPPAAL and MOBY/RT is delegated to practical work of the students in separate labs on real-time systems. There LEGO Mindstorm robots are used for implementing the systems. Once desirable real-time properties have been verified, the compiler from PLC-Automata to C is applied to generate code for the LEGO Mindstorms.

An alternative usage of the material of this book could be in (part of) a course on timed automata as follows:

Course based on timed automata	
Introduction	1
Timed automata	4
PLC-Automata	5.1–5.3 and 5.6
Automatic verification	6

Further information and additional material can be found on the webpage `http://csd.informatik.uni-oldenburg.de/rt-book`.

Acknowledgements

Our first inspiring contacts with real-time systems were in the context of the basic research project ProCoS (Provably Correct Systems) funded by the European Commission from 1989 to 1995. This project was planned by Dines Bjørner (Technical University of Denmark), Tony Hoare (Oxford University), and Hans Langmaack (University of Kiel). Its goal was to develop a mathematical basis for the development of embedded, real-time, computer systems.

Returning from a sabbatical at the University of Austin at Texas, Tony Hoare was impressed by the work of Robert S. Boyer and J Strother Moore on mechanical verification exemplified in a case study known as the "CLInc Stack". Talking to Dines Bjørner and Hans Langmaack, a project on the foundation of verification of many-layered systems was conceived: ProCoS. The different levels of abstraction studied in this project became known as the "ProCoS Tower". They comprise (informal) expectations, (formal) requirements, (formal) system specifications, programs (occam), machine code (for transputers), and circuit diagrams (netlists). During the project the case study of a gas burner was defined in collaboration with a Danish gas burner manufacturer.

At the project start in 1989 the first author of this book moved from Kiel to Oldenburg to take up a professorship in computing science at the University of Oldenburg and became one of the site leaders of ProCoS. He is very grateful for six rewarding years of research contacts with the members of the ProCoS project group, in particular Hans Langmaack, Tony Hoare, Dines Bjørner, Zhou Chaochen, He Jifeng, Jonathan Bowen, Michael R. Hansen, Anders P. Ravn, Hans Rischel, Kirsten M. Hansen, Martin Fränzle, Markus Müller-Olm, Stephan Rössig, and Michael Schenke. Two highlights evolved during the ProCoS project: the case study of the gas burner and the Duration Calculus, both featuring prominently in this book.

In the first years of ProCoS the second author of this book was a student of computing science and mathematics at Oldenburg. His first contact with the real-time systems of ProCoS was during his master thesis on "The production cell as a verified real-time system" – formalised using the Duration Calculus.

The next decisive step was the collaborative project UniForM (Universal Workbench for Formal Methods) together with Bernd Krieg-Brückner and Jan Peleska (University of Bremen) as well as Alexander Baer and Wolfgang Nowak (company Elpro AG in Berlin). One of the challenges of this project was to develop a formal method to support the real-time programming of tram control systems targeted at Programmable Logic Controllers. Motivated by this challenge the second author developed the concept of a PLC-Automaton, which serves for design specifications in this book.

Inspired by ProCoS and UniForM the research on specification and verification of real-time systems gained momentum at our group on "Correct System Design" at Oldenburg. In particular, we wish to thank Cheryl Kleuker, who contributed Constraint Diagrams, Jochen Hoenicke, who can spot even subtle errors in a minute, and Andreas Schäfer, who saw how to extend the Duration Calculus to cope with space and time. Under the guidance of Josef Tapken the tool MOBY/RT was developed to provide support for the theory presented in this book. We are particularly grateful to the following people who helped create this tool: Hans Fleischhack, Marc Lettrari, Michael Möller, Marco Oetken, Josef Tapken, and Tobe Toben.

The second author spent an extended research visit at the Aalborg University to work with the UPPAAL group on automatic verification and planning of timed automata. He would like to thank Kim Larsen, Gerd Behrmann, Alexandre David, Anders P. Ravn, Wang Yi, and Paul Pettersson for inspiring cooperation.

Both authors are pleased to acknowledge the research momentum gained by the Collaborative Research Center AVACS (Automatic Verification and Analysis of Complex Systems) which has been funded by the German Research Council (DFG) since 2004. AVACS groups at the universities of Oldenburg, Freiburg and Saarbrücken, as well as the Max-Planck Institute for Informatics in Saarbrücken, address automatic verification and analysis of real-time systems, hybrid systems, and systems of systems. In the research area of real-time systems we would like to thank our close colleagues Werner Damm, Bernd Becker, Reinhard Wilhelm, Johannes Faber, Roland Meyer, Ingo Brückner, Heike Wehrheim, Bernd Finkbeiner, Andreas Podelski, Andrey Rybalchenko, Viorica Sofroni-Stokkermans, Bernhard Nebel, Jörg Hoffmann, and Sebastian Kupferschmid. We also thank Willem-Paul de Roever for his support of this large-scale project and for many refreshing remarks and suggestions over the years.

Everyone who has written a book knows how difficult it is to find the time to work intensively on the manuscript. Very helpful in this respect was a sabbatical of the first author in the winter semester 2004/05 at ETH Zürich. Many thanks to my perfect hosts David Basin and Barbara Geiser. The first author would also like to thank Krzysztof R. Apt, with whom he wrote his first book, for setting a lucid example of how a book should look and for many pieces of invaluable advice during the past years.

We are very grateful to Michael Möller for creating a draft on which the cover design of this book is based. Last but not least we wish to thank David Tranah and his team from Cambridge University Press who have been very supportive throughout this book project.

List of symbols

$[\nu]$ (region)	159
$[\varphi]$ (region)	159
$\longrightarrow$ (followed-by)	98
$\longrightarrow_0$ (followed-by-initially)	99
$\xrightarrow{\theta}$ (leads-to)	99
$\xrightarrow{\alpha}$ (discrete transition)	140, 142
$\xrightarrow{t}$ (delay transition)	139, 142
$\xrightarrow{\leq\theta}$ (up-to)	100
$\xrightarrow{\leq\theta}_0$ (up-to-initially)	101
$\vdash$ (provability)	54
; (chop)	39
$\circ$ (relational composition)	140
$\cong$ (region equivalence)	157
$\overset{\mathrm{def}}{=}$ (equality by definition)	37
$\overset{\mathrm{def}}{\iff}$ (equivalence by definition)	9
$\Box$ (everywhere)	44
$\Diamond F$ (somewhere)	43
$\lceil P \rceil$ (almost everywhere)	43
$\lceil P \rceil^t$ (variant with duration)	43
$\lceil P \rceil^{\leq t}$ (variant with bound)	43
$\lceil\rceil$	43
$\mathcal{I}[\![P]\!]$	34
$\models$ (models)	45
$\models_0$ (models from 0)	46
$\models_1$	241
$\simeq$ (isomorphism)	147
A (approaching)	9
Act (set of actions)	136
$B_{?!}$ (action set)	136
Chan (set of channels)	136
Cl (closed)	9
Cr (cross)	9
$DNF(P)$	86
$\mathsf{Des\text{-}1}$ (requirement)	15
$\mathsf{Des\text{-}2}$ (requirement)	15
E (empty)	9
FA (finite alternation)	65
GVar (set of global variables)	31
Intv (set of closed intervals)	36
Lab (set of labels)	136
$\mathbb{N}$ (natural numbers)	294
O (open)	9
Obs (set of observables)	31
$Pred(\pi, t)$	218
Pref (prefix operator)	114
$\mathbb{Q}$ (rational numbers)	294
$\mathbb{R}$ (real numbers)	294
$R(\mathbb{X}, V)$ (reset operations)	167
Req (requirement)	14
Time (time domain)	4
Track	9
Val (set of valuations)	32
$\mathbb{X}$ (set of clocks)	136
$\mathbb{Z}$ (integer)	294
$a!$ (output action)	136
$a?$ (input action)	136
appr	9
$bounds(\pi)$	215
chan (underlying channel)	136
$\mathsf{chan}\, b \bullet \mathcal{A}$ (restriction)	146
cross	9
$delay[\nu]$ (delay operation)	163
empty	9
$first(\pi)$	220
$free$ (set of free global variables)	40
g (gate)	9
$kern(L)$ (kernel)	88
$\langle \ell, \nu \rangle$	140

$\vec{\ell}$ (control vector)	148, 173
$\langle \ell, \nu \rangle, t$ (time-stamped config.)	142
max (maximum)	298
min (minimum)	298
obs (observable)	4
$\langle\, r_1, \ldots, r_n \,\rangle$ (reset operations)	167
$\vec{r}$ (reset operations)	167
$\Phi(V)$ (integer constraint)	166
$\Phi(\mathbb{X})$ (clock constraint)	136
$\Phi(\mathbb{X}, V)$ (guards)	167
$\Psi(V)$ (integer expression)	166
$\overline{\alpha}$ (complementary action)	136
λ (label)	136
ν (clock valuation)	138
$\nu + t$ (time shift)	139
$\nu[Y := t]$ (modification)	139
τ (internal action)	136
θ (DC term)	36
ξ (computation path)	142
$\mathcal{C}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ (network)	173
$\mathcal{D}$ (data type)	4
$\mathcal{I}$ (interpretation)	32
$\mathcal{N}$ (network to timed automata)	148
$\mathcal{P}$ (power set)	295
$\mathcal{R}$ (structure of real numbers)	61
$\mathcal{R}(\mathcal{A})$ (region automaton)	160
$\mathcal{X}$ (formula variable)	72

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.

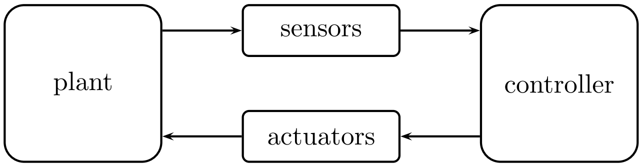

Fig. 1.1. Real-time system

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.

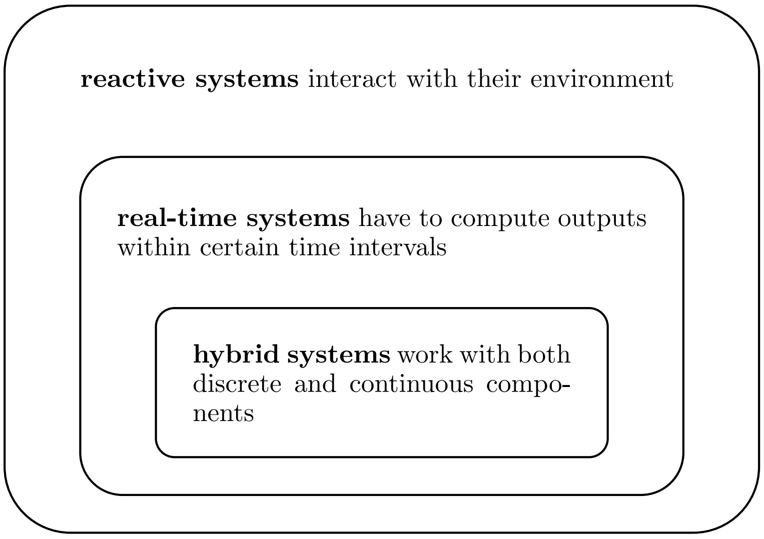
reactive systems interact with their environment
real-time systems have to compute outputs within certain time intervals
hybrid systems work with both discrete and continuous components

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.

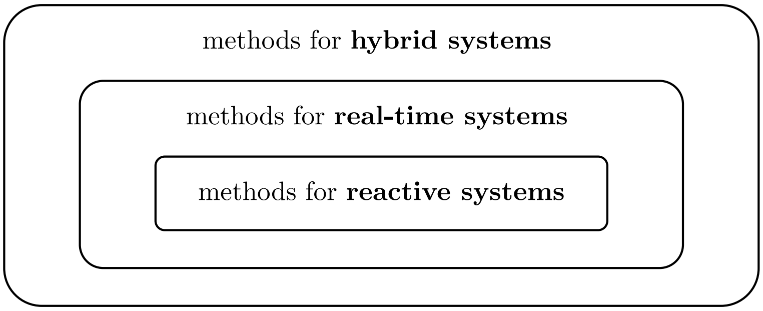
methods for **hybrid systems**
methods for **real-time systems**
methods for **reactive systems**

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

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

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

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

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

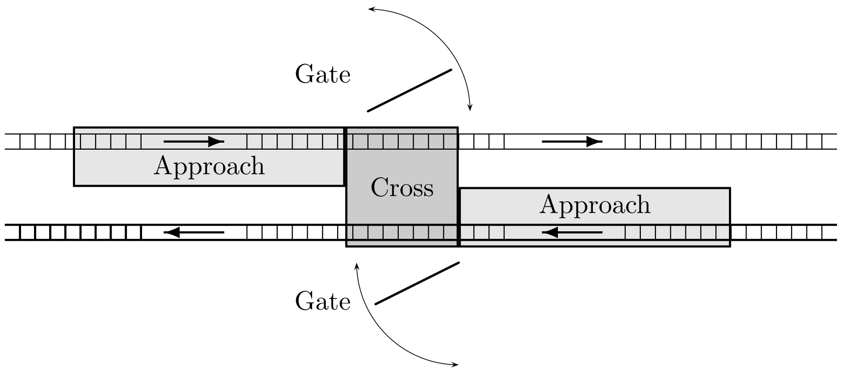

Fig. 1.4. Generalised railroad crossing

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:

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

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:

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

where $\overset{\text{def}}{\iff}$ 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.

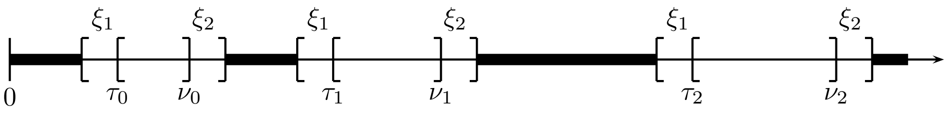

Fig. 1.5. Utility requirement

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 $t - \xi_2 \le \tilde{t} \le t + \xi_1$. This is expressed by the following formula:

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

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 $\mathsf{Utility}$ 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: $\mathsf{Init} \overset{\text{def}}{\iff} E(0)$.

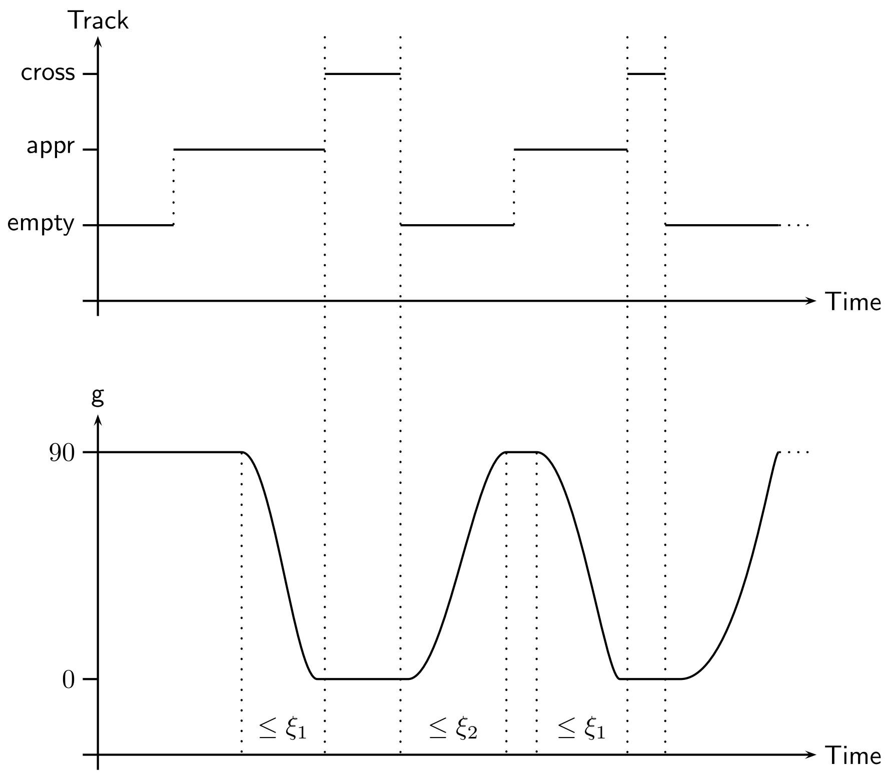

Fig. 1.6. An admissible interpretation of the observables Track and g

- Trains cannot enter the crossing without approaching it:

$$\begin{aligned}\mathsf{E\text{-}to\text{-}Cr} \;\overset{\text{def}}{\iff}\; & \forall b, e \in \mathsf{Time} \bullet (b \leq e \wedge E(b) \wedge Cr(e)) \\ & \implies \exists t \in \mathsf{Time} \bullet b < t < e \wedge A(t).\end{aligned}$$

- Approaching trains eventually cross:

$$\begin{aligned}\mathsf{A\text{-}to\text{-}E} \;\overset{\text{def}}{\iff}\; & \forall b, e \in \mathsf{Time} \bullet (b \leq e \wedge A(b) \wedge E(e)) \\ & \implies \exists t \in \mathsf{Time} \bullet b < t < e \wedge Cr(t).\end{aligned}$$

Some assumptions about the speed of the approaching trains are also needed. If a train could approach the crossing arbitrarily fast, a typical reaction time of half a minute for the gates to close would not suffice. We assume that the fastest train will take a time of ρ to reach the crossing after being detected in the approaching area. Here $\rho > 0$ is another time parameter. On the other hand, trains which are arbitrarily slow in the approaching area are not acceptable in the presence of the utility requirement. Therefore we assume that trains need not more than ρ' to pass through the approaching area.

- Fastest train:

$$\mathsf{T\text{-}Fast} \overset{\text{def}}{\iff} \forall c, d \in \mathsf{Time} \bullet (c < d \wedge E(c) \wedge Cr(d)) \Longrightarrow d - c \geq \rho.$$

- Slowest train:

$$\mathsf{T\text{-}Slow} \overset{\text{def}}{\iff} \forall c \in \mathsf{Time} \bullet A(c) \Longrightarrow (\exists d \in \mathsf{Time} \bullet c < d < c + \rho' \wedge \neg A(d)).$$

1.3.3 Design

For the design of the controller we stipulate that the gate is closed at most ξ_1 seconds after detection of an approaching train:

$$\mathsf{Des\text{-}G} \overset{\text{def}}{\iff} \forall c, d \in \mathsf{Time} \bullet d - c \geq \xi_1 \wedge (\forall t \in \mathsf{Time} \bullet c < t < d \Longrightarrow \neg E(t)) \Longrightarrow Cl(d).$$

Under the assumptions

$$\mathsf{Asm} \overset{\text{def}}{\iff} \mathsf{Init} \wedge \mathsf{T\text{-}Fast} \wedge \rho \geq \xi_1$$

we can then prove that the following implication holds:

$$(\mathsf{Asm} \wedge \mathsf{Des\text{-}G}) \Longrightarrow \mathsf{Safety}.$$

Thus for all interpretations of Track and g satisfying Asm and $\mathsf{Des\text{-}G}$, the safety requirement Safety holds.

Proof:
See Exercise 1.3. □

1.4 Gas burner

This case study was introduced in [RRH93, HHF$^+$94] during the EU project ProCoS (Provably Correct Systems, 1989–95, [BHL$^+$96]). The physical components of the plant are shown in Figure 1.7.

1.4.1 The problem

The desired functionality of the gas burner is as follows:

- If the thermostat signals to switch on the heating the gas valve opens and the burner tries to ignite it for a short period of time.

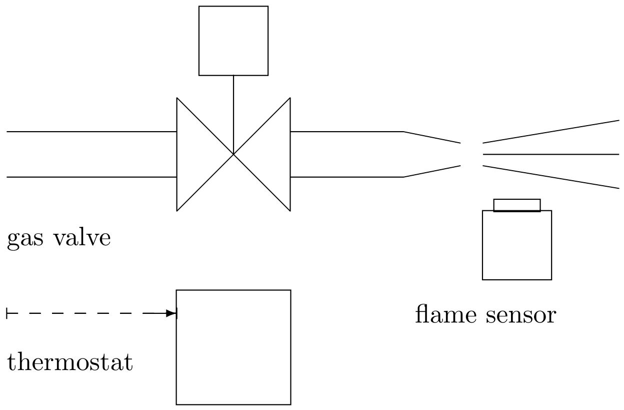

Fig. 1.7. Gas burner

- If the thermostat signals to switch off the heating the gas valve closes.

Important is the following *safety-critical* aspect of the gas burner. If gas effuses without a burning flame in front of the gas valve the concentration of unburned gas can reach critical limits and thus cause an explosion. This has to be avoided. To this end, the following real-time constraint on the system is introduced:

- For each time interval with a duration of at least 60 seconds the (accumulated) duration of gas leaks is at most 5% of the overall duration.

Note that this requirement does not exclude short gas leaks because they are unavoidable before ignition. If the system satisfies this requirement the gas burner is safe.

1.4.2 Formalisation

We concentrate on the safety aspect of the gas burner and introduce two Boolean observables: G describes whether the gas valve is open, and F whether the flame is burning as detected by the flame sensor.

$$\begin{aligned} G &: \mathsf{Time} \longrightarrow \{0,1\} \\ F &: \mathsf{Time} \longrightarrow \{0,1\}. \end{aligned}$$

The safety-critical state L describes when gas *leaks*, i.e. when G holds but F does not. It is formalised by the Boolean expression $L \overset{\text{def}}{\Longleftrightarrow} G \wedge \neg F$, which is time dependent just as G and F are:

$$L : \mathsf{Time} \longrightarrow \{0, 1\}.$$

Figure 1.8 exhibits an example of interpretations for F and G and the resulting value for L.

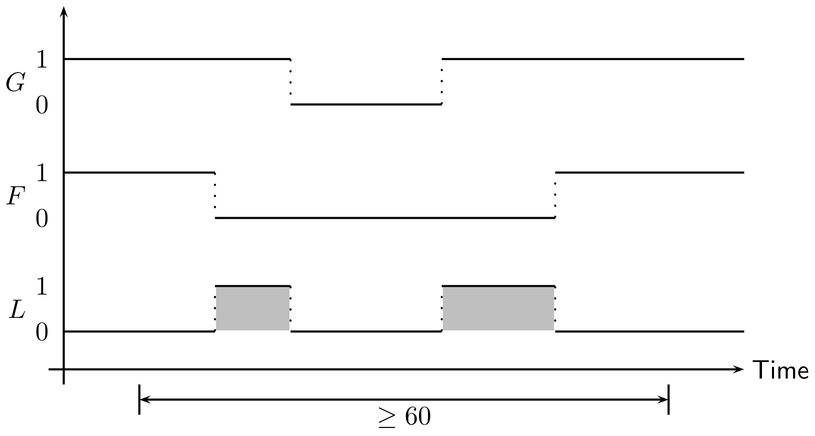

Fig. 1.8. Interpretations for F, G, and L

The real-time requirement is that for each time interval of at least 60 seconds duration the shaded periods do not exceed 5%, i.e. one-twentieth of that duration. To measure in a given interval $[b, e]$ the sum of the durations of all subintervals in which $L(t) = 1$ holds, we use the *integral notation*

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

Here L is considered as a function from real numbers to real numbers, which is integrable under suitable assumptions. The requirement can now be formalised as follows:

$$\mathsf{Req} \overset{\text{def}}{\Longleftrightarrow} \forall b, e \in \mathsf{Time} \bullet \left(e - b \geq 60 \Longrightarrow \int_b^e L(t)dt \leq \frac{e-b}{20} \right). \tag{1.7}$$

Looking at this high-level requirement it is difficult to see how to construct a controller that guarantees it.

1.4.3 Design

As a step towards a controller we make the design decision to introduce two real-time constraints that seem easier to implement and that together imply the requirement Req.

(i) The controller can stop each leak *within a second*:

$$\begin{aligned}\mathsf{Des\text{-}1} \overset{\text{def}}{\iff} \forall b, e \in \mathsf{Time} \bullet\ & (\forall t \in \mathsf{Time}\ \bullet b \leq t \leq e \Longrightarrow L(t)) \\ & \Longrightarrow e - b \leq 1.\end{aligned}$$

This constraint restricts the duration of each leak state to at most one second.

(ii) After each leak the controller *waits for 30 seconds* before opening the gas valve again:

$$\begin{aligned}\mathsf{Des\text{-}2} \overset{\text{def}}{\iff} \forall b, e \in \mathsf{Time} \bullet\ & (L(b) \wedge L(e) \wedge \\ & \exists t \in \mathsf{Time} \bullet (b < t < e \wedge \neg L(t))) \\ & \Longrightarrow e - b \geq 30.\end{aligned}$$

This constraint requests a *distance* of at least 30 seconds between any two subsequent leak states. This is illustrated in Figure 1.9.

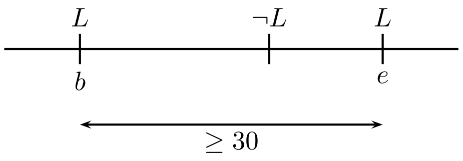

Fig. 1.9. Real-time constraint $\mathsf{Des\text{-}2}$

From these design constraints it is possible to prove the desired requirement because the following implication holds:

$$(\mathsf{Des\text{-}1} \wedge \mathsf{Des\text{-}2}) \Longrightarrow \mathsf{Req},$$

i.e. for all interpretations of G and F satisfying $\mathsf{Des\text{-}1}$ and $\mathsf{Des\text{-}2}$, the safety requirement Req holds.

1.5 Aims of this book

Using predicate logic as a specification language for real-time systems has several disadvantages. First, as we have seen in the examples above, we have to spell out explicitly all quantifications over time. Second, there is no support for an automatic verification of properties that one might want to prove about such specifications. Third, there is no obvious way to implement a real-time system once it is specified in predicate logic.

To overcome these disadvantages we shall consider three dedicated formal specification languages for real-time systems: Duration Calculus, timed automata, and PLC-Automata.

1.5.1 Duration Calculus

The *Duration Calculus* (abbreviated DC) was introduced by Zhou Chaochen in collaboration with M.R. Hansen, C.A.R. Hoare, A.P. Ravn, and H. Rischel. The DC is a temporal logic and calculus for describing and reasoning about properties that time-dependent observables satisfy over time intervals. In particular, safety properties, bounded response, and duration properties (hence the name of the calculus) can be expressed in DC.

Example 1.1
The safety requirement Req for the gas burner that we formalised in Section 1.4.2 using predicate logic can be expressed in DC more concisely by the duration formula

$$\Box\left(\ell \geq 60 \Longrightarrow \textstyle\int L \leq \dfrac{\ell}{20}\right).$$

It states that for all observation intervals ($\Box$) of length at least 60 seconds ($\ell \geq 60$) the accumulated duration of a gas leak $\left(\int L\right)$ is at most 5%, i.e. one-twentieth of the length of the interval $\left(\leq \frac{\ell}{20}\right)$. Note that in contrast to the formula in predicate logic this DC formula avoids any explicit quantification over time points. ■

An advantage of DC is that it enables us to express a high-level declarative view of real-time systems without implementation bias. We shall therefore use DC as a specification language for system requirements. The price to pay is that for the continuous-time domain the satisfiability problem of the DC is in general undecidable. Thus we cannot hope for automatic verification procedures for the full DC. Also direct tool support for the DC is at present rather limited.