Communicating Embedded Systems

Software and Design

Edited by
Claude Jard and Olivier H. Roux

iSTE

Communicating Embedded Systems

Communicating Embedded Systems

Software and Design

Formal Methods

Edited by
Claude Jard
Olivier H. Roux

iSTE

First published 2008 in France by Hermes Science/Lavoisier entitled: *Approches formelles des systèmes embarqués communicants* © LAVOISIER 2008
First published 2010 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the CLA. Enquiries concerning reproduction outside these terms should be sent to the publishers at the undermentioned address:

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

© ISTE Ltd 2010

The rights of Claude Jard and Olivier H. Roux to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

Library of Congress Cataloging-in-Publication Data

Approches formelles des systèmes embarqués communicants. English
Communicating embedded systems : software and design : formal methods / edited by Claude Jard, Olivier H. Roux.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-84821-143-8
1. Embedded computer systems--Programming. 2. Embedded computer systems--Design and construction. 3. Computer software--Development. 4. Formal methods (Computer science) I. Jard, Claude. II. Roux, Olivier H. III. Title.
TK7895.E42A387 2009
621.39'2--dc22

2009026282

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-84821-143-8

Edited and formatted by Aptara Corporation, New Delhi, India
Printed and bound in Great Britain by CPI Antony Rowe, Chippenham and Eastbourne

Contents

Preface xi
Claude JARD and Olivier H. ROUX

Chapter 1. Models for Real-Time Embedded Systems 1
Didier LIME, Olivier H. ROUX and Jiří SRBA

1.1. Introduction 1
1.1.1. Model-checking and control problems 2
1.1.2. Timed models 3
1.2. Notations, languages and timed transition systems 5
1.3. Timed models 8
1.3.1. Timed Automata 8
1.3.2. Time Petri nets 10
1.3.2.1. T-time Petri nets 12
1.3.2.2. Timed-arc petri nets 15
1.3.3. Compared expressiveness of several classes of timed models . . . 19
1.3.3.1. Bisimulation and expressiveness of timed models 19
1.3.3.2. Compared expressiveness of different classes of TPN 20
1.3.3.3. Compared expressiveness of TA, TPN, and TAPN 21
1.4. Models with stopwatches 23
1.4.1. Formal models for scheduling aspects 23
1.4.1.1. Automata and scheduling 23
1.4.1.2. Time Petri nets and scheduling 24
1.4.2. Stopwatch automata 25
1.4.3. Scheduling time Petri nets 26
1.4.4. Decidability results for stopwatch models 31
1.5. Conclusion 31
1.6. Bibliography 31

Chapter 2. Timed Model-Checking . . . 39
Béatrice BÉRARD

2.1. Introduction . . . 39
2.2. Timed models . . . 40
2.2.1. Timed transition system . . . 40
2.2.2. Timed automata . . . 41
2.2.3. Other models . . . 44
2.3. Timed logics . . . 46
2.3.1. Temporal logics CTL and LTL . . . 46
2.3.2. Timed extensions . . . 48
2.3.2.1. Timed CTL . . . 48
2.3.2.2. Timed LTL . . . 50
2.4. Timed model-checking . . . 51
2.4.1. Model-checking LTL and CTL (untimed case) . . . 51
2.4.2. Region automaton . . . 53
2.4.3. Model-checking TCTL . . . 56
2.4.4. Model-checking MTL . . . 58
2.4.5. Efficient model-checking . . . 59
2.4.6. Model-checking in practice . . . 60
2.5. Conclusion . . . 61
2.6. Bibliography . . . 61

Chapter 3. Control of Timed Systems . . . 67
Franck CASSEZ and Nicolas MARKEY

3.1. Introduction . . . 67
3.1.1. Verification of timed systems . . . 67
3.1.2. The controller synthesis problem . . . 68
3.1.3. From control to game . . . 69
3.1.4. Game objectives . . . 70
3.1.5. Varieties of untimed games . . . 71
3.2. Timed games . . . 72
3.2.1. Timed game automata . . . 72
3.2.2. Strategies and course of the game . . . 73
3.2.2.1. The course of a timed game . . . 73
3.2.2.2. Strategies . . . 74
3.3. Computation of winning states and strategies . . . 76
3.3.1. Controllable predecessors . . . 77
3.3.2. Symbolic operators . . . 79
3.3.3. Symbolic computation of winning states . . . 79
3.3.4. Synthesis of winning strategies . . . 80

3.4. Zeno strategies . . . 82
3.5. Implementability . . . 82
3.5.1. Hybrid automata . . . 83
3.5.2. On the existence of non-implementable continuous controllers . . 84
3.5.3. Recent results and open problems . . . 85
3.6. Specification of control objectives . . . 85
3.7. Optimal control . . . 87
3.7.1. TA with costs . . . 87
3.7.2. Optimal cost in timed games . . . 89
3.7.3. Computation of the optimal cost . . . 90
3.7.4. Recent results and open problems . . . 92
3.8. Efficient algorithms for controller synthesis . . . 92
3.8.1. On-the-fly algorithms . . . 93
3.8.2. Recent results and open problems . . . 95
3.9. Partial observation . . . 96
3.10. Changing game rules... . . . 97
3.11. Bibliography . . . 98

Chapter 4. Fault Diagnosis of Timed Systems . . . 107
Franck CASSEZ and Stavros TRIPAKIS

4.1. Introduction . . . 107
4.2. Notations . . . 109
4.2.1. Timed words and timed languages . . . 109
4.2.2. Timed automata . . . 110
4.2.3. Region graph of a TA . . . 111
4.2.4. Product of TA . . . 111
4.2.5. Timed automata with faults . . . 112
4.3. Fault diagnosis problems . . . 113
4.3.1. Diagnoser . . . 113
4.3.2. The problems . . . 114
4.3.3. Necessary and sufficient condition for diagnosability . . . 115
4.4. Fault diagnosis for discrete event systems . . . 115
4.4.1. Discrete event systems for fault diagnosis . . . 115
4.4.2. Checking Δ-diagnosability and diagnosability . . . 116
4.4.2.1. Checking Δ-diagnosability . . . 116
4.4.2.2. Checking diagnosability . . . 117
4.4.3. Computation of the maximum delay . . . 120
4.4.4. Synthesis of a diagnoser . . . 121
4.5. Fault diagnosis for timed systems . . . 122
4.5.1. Checking Δ-diagnosability . . . 122
4.5.2. Checking diagnosability . . . 123

4.5.3. Computation of the maximal delay 125
4.5.4. Synthesis of a diagnoser 126
4.5.5. Fault diagnosis with deterministic timed automata 127
4.6. Other results and open problems 136
4.7. Bibliography 136

Chapter 5. Quantitative Verification of Markov Chains 139
Susanna DONATELLI and Serge HADDAD

5.1. Introduction 139
5.2. Performance evaluation of Markov models 140
5.2.1. A stochastic model for discrete events systems 140
5.2.2. Discrete time Markov chains 143
5.2.3. Continuous time Markov chain 146
5.3. Verification of discrete time Markov chain 148
5.3.1. Temporal logics for Markov chains 148
5.3.2. Verification of PCTL formulae 149
5.3.3. Aggregation of Markov chains 151
5.3.4. Verification of PLTL formulae 154
5.3.5. Verification of $PCTL^*$ 157
5.4. Verification of continuous time Markov chain 157
5.4.1. Limitations of standard performance indices 157
5.4.2. A temporal logics for continuous time Markov chains 158
5.4.3. Verification algorithm 159
5.5. State of the art in the quantitative evaluation of Markov chains 160
5.6. Bibliography 162

Chapter 6. Tools for Model-Checking Timed Systems 165
Alexandre DAVID, Gerd BEHRMANN, Peter BULYCHEV, Joakim BYG, Thomas CHATAIN, Kim G. LARSEN, Paul PETTERSSON, Jacob Illum RASMUSSEN, Jiří SRBA, Wang YI, Kenneth Y. JOERGENSEN, Didier LIME, Morgan MAGNIN, Olivier H. ROUX and Louis-Marie TRAONOUEZ

6.1. Introduction 165
6.2. UPPAAL 166
6.2.1. Timed automata and symbolic exploration 166
6.2.1.1. Example 169
6.2.2. Queries 170
6.2.3. Architecture of the tool 172
6.2.4. Reachability pipeline 173
6.2.5. Liveness pipeline 175
6.2.6. Leadsto pipeline 176
6.2.7. Active clock reduction 176

6.2.8. Space reduction techniques 177
6.2.8.1. Avoid storing all states 177
6.2.8.2. Sharing data 178
6.2.8.3. Minimal graph 178
6.2.8.4. Symmetry reduction 179
6.2.9. Approximation techniques 180
6.2.9.1. Over-approximation: convex-hull 180
6.2.9.2. Under-approximation: bit-state hashing 180
6.2.10. Extensions 181
6.2.10.1. Robust reachability 181
6.2.10.2. Merging DBMs 181
6.2.10.3. Stopwatches 181
6.2.10.4. Supremum values 181
6.2.10.5. Other extensions 181
6.3. UPPAAL-CORA 182
6.3.1. Priced timed automata 182
6.3.2. Example 184
6.4. UPPAAL-TIGA 185
6.4.1. Timed game automata 185
6.4.2. Reachability pipeline 187
6.4.3. Time optimality 188
6.4.4. Cooperative strategies 189
6.4.5. Timed games with Büchi objectives 190
6.4.6. Timed games with partial observability 192
6.4.6.1. Algorithm 194
6.4.6.2. Implementation 194
6.4.7. Simulation checking 196
6.4.7.1. Algorithm 197
6.5. TAPAAL 199
6.5.1. Introduction 199
6.5.2. Definition of timed-arc Petri nets used in TAPAAL 200
6.5.3. TAPAAL logic 203
6.5.4. Tool details 204
6.6. ROMÉO: a tool for the analysis of timed extensions of Petri nets 205
6.6.1. Models 206
6.6.1.1. Time Petri nets 206
6.6.1.2. Petri Nets with stopwatches 208
6.6.1.3. Parametric Petri nets with stopwatches 210
6.6.2. Global architecture 210
6.6.3. Systems modeling 211
6.6.4. Verification of properties 211

6.6.4.1. On-line *model checking* 211
6.6.4.2. Off-line *model checking* 213
6.6.5. Using ROMÉO in an example 214
6.7. Bibliography 217

Chapter 7. Tools for the Analysis of Hybrid Models 227
Thao DANG, Goran FREHSE, Antoine GIRARD and Colas LE GUERNIC

7.1. Introduction 227
7.2. Hybrid automata and reachability 228
7.3. Linear hybrid automata 232
7.4. Piecewise affine hybrid systems 234
7.4.1. Time discretization 234
7.4.1.1. Autonomous dynamics 234
7.4.1.2. Dynamics with inputs 236
7.4.2. Scaling up reachability computations 237
7.4.2.1. Reachability using zonotopes 237
7.4.2.2. Efficient implementation for LTI systems 239
7.4.2.3. Dealing with the discrete transitions 239
7.5. Hybridization techniques for reachability computations 241
7.5.1. Approximation with linear hybrid automata 241
7.5.2. Hybridization of nonlinear continuous system 243
7.5.2.1. Properties of the approximate reachable set 244
7.5.2.2. Approximation by hybrid systems with piecewise affine dynamics 245
7.5.3. Hybridization and refinement 246
7.6. Bibliography 249

List of Authors 253

Index 259

Preface

This book stems from the global scientific priority put on the development of the research on communicating embedded systems. A worldwide scientific community is motivated by this theme and taking care of the development and promotion of formal approaches based on mathematical models and implemented by computer-automated tools. The main objective is to rise to the ever-growing challenge of mastering the quality of upcoming computer systems.

The computer systems concerned are:

– critical: a strong demand for quality comes from their handling of critical functions or a large-scale deployment;

– complex: their correct functioning depends on the careful use of resources that are mutually dependent over time. So their analysis on the basis of a formal model appears to be the most solid scientific approach.

This motivation is not new as the formal methods community has already produced a great number of results, mainly dedicated to the tasks of formal specification, verification and proof, and test synthesis. Several studies have been conducted with industrial partners, and a few of them have finally initiated some industrial software development processes. However, it remains mostly insufficient with respect to the increasing computer-related risks. Some progress is expected on the development and promotion of formal methods used in concrete computer-related problems and objects, as well as in the systematic research of new application fields and methods. Beyond the "traditional" application of formal methods to the conception of safe software, a significant impact can already be observed on areas of compiling and synthesis of controllers, diagnosis and supervision, system engineering, or security and safety engineering.

Chapter written by Claude JARD and Olivier H. ROUX.

Considering the widely acknowledged importance of the human and economic interests at stake with the future embedded systems (encompassing many economic sectors that have yet to be discovered), it is necessary to take particular interest in widely diffusing the technical advances of the domain. This is precisely the aim of this book arranged in an unified manner, such as the elements related to the current problems of embedded systems, the presentation of the achievements of new research directions on models and their use, and the presentation of available software tools. The contributing authors are renowned specialists in their respective fields. We have taken care of the coordination of the whole book, as leaders of the French national action AFSEC ("Formal Approaches for Communicating Embedded Systems"), supported by the CNRS.

Chapter 1

Models for Real-Time Embedded Systems

1.1. Introduction

The class of *real-time embedded* systems contains software components that control an application by reacting to stimuli received from a changing environment. Therefore, they are often referred to as *reactive* systems. The reaction time of these systems must be small enough to cope with the internal dynamics of the controlled or monitored application. They must thus obey the strong timing requirements, and it is crucial to ensure their correctness from both the functional and temporal points of view.

Real-time applications are often regarded as safety critical because their failures may either involve substantial financial losses or endanger human lives. It is also important to detect any error at an early stage to minimize the costs involved in its correction. This issue can be addressed by several approaches. Let us consider the following two approaches: in the first approach, from the application requirements, a model for the application and its expected properties are derived. Then a controller is proposed, using expert knowledge, to restrict the application behavior in order to satisfy the given properties. The next step is to validate the proposed controller using different techniques such as testing, theorem proving, formal verification, etc. If the controller is not acceptable, then it has to be reworked on the basis of the knowledge gained from the reported failures or counter-examples. In the second approach, the controller will be synthesized automatically from the formal model of the application. If such a

---

Chapter written by Didier LIME, Olivier H. ROUX and Jiří SRBA.

controller exists, which may not be the case, for example, if the requirements are incoherent, then a method to implement should be derived. Alternatively, if the controller model cannot be implemented, the problem has to be reworked from the beginning.

In this chapter, we will address the formal methods relating to these two approaches and those based on state-transition models. The models presented here deal with instantaneous actions and real-valued variables. The former approach usually models relevant events in the systems such as those relative to the command of the actuators or to the measures of sensors and the latter approach models durations of processes. These models thus describe subclasses of real-time embedded systems.

1.1.1. *Model-checking and control problems*

The requirements of a computer application are usually given as a natural language document. Therefore, they are subject to the interpretation of the persons who will derive the specification from them, which may result in many errors. Moreover, it is impossible for a conceptor to understand all possible interactions of the components of any reasonably sized application.

Formal methods aim at providing a mathematical framework for a clear, non-ambiguous and precise description of the systems and programs we want to develop. In this framework, the system is described by a labeled transition system or any model abstracting such a systems, e.g. automata, Petri nets, process algebra, etc. The specification may be either described in the same manner or as a property expressed in a suitable logic, such as linear temporal logics (LTL) or tree-based temporal logics (CTL), or even their timed extensions. A classical example of a real-time property is the *bounded response* property which requires that for the execution of the system, whenever some predicate P_0 on the states of the system becomes true, then some other predicate P_1 will become true, within n time units.

Among formal methods, *model-checking* is an automatic procedure that verifies whether a system satisfies a given temporal logic property. It usually works by exploring parts of the set of possible states of the system.

Controller synthesis consists of automatically synthesizing a program (the controller) that leads the interaction mechanisms with the environment (the application) and guarantees a safe and correct behavior of the coupled system. The set of actions of the system is partitioned into two disjoint subsets: a set of controllable actions and a set of uncontrollable actions. They usually correspond to the commands to actuators and the measures of sensors, respectively. The controller of the application may act on the controllable actions (and only on them) and the resulting system is said to be *closed* (as opposed to the system without the controller, which is said to be *open*).

Let us now compare the model-checking and control problems more precisely. In the model-checking problem, an already *closed* system is considered; let us call it S_{closed}. This system may be best viewed as an application S to which a controller C has been added. The expected correctness property is given by a formula φ in a suitable logic. The model-checking problem consists of deciding the question "does S_{closed} satisfy φ?" or equivalently "does S controlled by C satisfy φ?" which we will explain by:

$$\text{Does } S \parallel C \models \varphi? \qquad [1.1]$$

The control problem is more general: it asks us to decide whether there exists a controller C of S such that the expected property φ is verified by S_{closed}:

$$\text{Does there exist } C \text{ such that } S \parallel C \models \varphi? \qquad [1.2]$$

This problem is generally more complicated than the classical model-checking problem. To find an answer to problem [1.2], the search-space of admissible controllers has to be defined first: for example, we might want a controller with a bounded memory (modeled, e.g. as a *finite automaton* or a *Petri net*). Moreover, the controller must be restrictive enough to enforce the expected safety properties and also allow enough to avoid an excessive limitation of the behaviors of the system. Finally, there is another problem related to the control problem: the *controller synthesis* problem asks us to effectively compute the controller C.

1.1.2. *Timed models*

Considering an expected property of the system, a suitable model can be defined as an abstraction of the system such that the answer to the considered problem (verification, control, etc.) is same for both the system and its model. This implies that a model is relevant only for a given set of properties and need not perfectly match the real behaviors of the system. However, there is usually an opposition between the *expressivity* of a class of models, i.e. its ability to faithfully account for a number of characteristics of the system, and the *simplicity* in its analysis, i.e. decidability and algorithmic complexity of the verification or control problems.

The choice of an expressive-enough model avoids the pessimism inherent to the over-approximation of the real behavior of the system. Thus, a coarse model may show behaviors that are actually prevented in the real system by some device not taken into account by the model. This pessimism may lead to the conclusion that some safety property is false on the model, whereas it is actually true. It may also lead to the generation of a state-space with no finite abstraction, whereas it would have been finite with a finer model. In particular, the state-space of an untimed model may have no finite abstraction, whereas adding timing information to the model may restrict the

behaviors to make a sufficient number of discrete states bounded and thus provide a finite abstraction.

Furthermore, in a number of applications, the requirements contain some properties that explicitly refer to durations between the two events. To account for such properties, a suitable model must embed some timing information rather than just the logical succession of events (logical time) expressed by models like finite automata or Petri nets. It must then model the dynamics of the application and/or the reactivity of the control system and provide *quantitative* timing information, thus enabling the verification of properties such as response times and making more precise failure causes to appear.

The models

Very general (and low-level) models such as transition systems (timed or untimed) are not directly usable for verification or control because they can represent any system without any restriction, which will usually imply the non-existence of finite representations. Therefore, we focus on higher level models, which can be analyzed using finite abstractions, but their semantics are given as transition systems.

In the untimed case, the simplest model is certainly a non-deterministic finite automaton, i.e. a labeled transition system with a finite number of states. Two other mainstream models are Petri nets and process algebras like *calculus of communicating systems* (CCS) [MIL 89]. All these models can be described in a finite way, even if their unfloodings in transition systems are usually infinite.

In the timed case, these models have been extended with timing information. Among these models, let us cite timed automata (TA) [ALU 94], extending finite automata, timed Petri nets [RAM 74], time Petri nets (TPNs) [MER 74], timed-arc Petri nets (TAPNs) [BOL 90, HAN 93], and timed process algebra such as TCCS [YI 91], one of the extensions of CCS. In this chapter, we focus on these "timed" models for which time is handled explicitly and, more precisely, on TA, TPNs and TAPNs.

Finally, the timed models can be further extended by using *stopwatches* to account for time elapsing, instead of *clocks*. These models are suspension/resumption of actions. The derivatives of the continuous variables relative to stopwatches can take only two values expressing either progression (1) or suspension (0). These models belong to the class of hybrid systems and they can model useful patterns such as pre-emptive scheduling in real-time systems.

Dense time vs. discrete time

Timed formal models account for the evolution of a system by both discrete instantaneous actions and time elapsing. The latter may be modeled either as a variable

valued over a dense domain (e.g. $\mathbb{R}$) or over a discrete domain (e.g. $\mathbb{N}$). This leads to the models being called "dense time" and "discrete time,' respectively'. The choice between the two may be motivated by the following pros and cons:

– real-life applications evolve along the physical time, which is best thought of as continuous and dense. Therefore, considering a discrete time leads to an under-approximation of the real behaviors of the application. However, in view of building a controller, the application is only monitored at specific moments in time (sampling or sporadic observations);

– the controller is made of a set of tasks running on one or several processors, whose physical time is discrete. Using a dense-time model for the controller might thus add behaviors not possible in the real system (depending on the time granularity of the processors). However, it is perfectly possible to model a discrete behavior using a dense-time model.

More pragmatic considerations on computational complexity or decidability of the verification and control problems may also favor the choice of models:

– state-space computation of dense-time models can be done using symbolic methods which gather finite or infinite number of states into equivalence classes, thus mitigating the combinatorial explosion commonly found with discrete-time models, when explicitly enumerating the states. However, efficient data structures exist on the basis of extensions of *binary decision diagrams* (BDD) that can partly handle the combinatorial explosion of the number of states;

– for some models, such as those featuring stopwatches for instance, the reachability of a given state is an undecidable problem when a dense time is considered; then there exists no algorithm solving the verification problem or simply computing the state-space, but only semi-algorithms. However, using a discrete time and bounded time constraints, the reachability problem (and many others) is decidable for those models.

1.2. Notations, languages and timed transition systems

General notations

– the set $\mathbb{B}$ contains the Boolean values tt and ff;

– $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ denote the sets of natural numbers, of rational numbers, and of real numbers, respectively;

– $\mathbb{R}_{\geq 0}$ is a set of non-negative real numbers and $\mathbb{R}_{>0} = \mathbb{R}_{\geq 0} \setminus \{0\}$ is a set of positive real numbers; we will also use these subscripts for $\mathbb{N}$ and $\mathbb{Q}$;

– for $n \in \mathbb{N}$, $\mathbb{R}^n$ is the space of real vectors of dimension n;

– for any finite set E, we denote the cardinality of E by $|E|$;

– B^A denotes the set of applications from A to B. If A is finite and $|A| = n$, an element in B^A is also a vector in B^n. The usual operations $+, -, <, \leq, >, \geq$, and $=$ are extended component-wise to vectors in A^n, with $A \in \{\mathbb{N}, \mathbb{Q}, \mathbb{R}\}$;

– a *valuation* ν on a set of variables X is an element of $\mathbb{R}_{\geq 0}^X$. For $\nu \in \mathbb{R}_{\geq 0}^X$ and $d \in \mathbb{R}_{\geq 0}$, $\nu + d$ denotes the valuation $(\nu + d)(x) = \nu(x) + d$, and for $X' \subseteq X$, $\nu[X' \mapsto 0]$ denotes the valuation ν' with $\nu'(x) = 0$ if $x \in X'$ and $\nu'(x) = \nu(x)$ otherwise. The symbol **0** denotes the null valuation such that $\forall x \in X, \nu(x) = 0$;

– let X be a set of variables. An *atomic constraint* on X is a formula of the form $x \sim c$ with $x \in X$, $c \in \mathbb{Q}_{\geq 0}$ and $\sim \in \{<, \leq, \geq, >\}$. $\mathcal{C}(X)$ denotes the set of *constraints* on X made from the conjunction of atomic constraints on X. Moreover, we denote by $\mathcal{C}_{dbm}(X)$ the set of conjunctions of terms of the form $x - x' \sim c$ or $x \sim c$, with $x, x' \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{Q}$;

– for a constraint $\varphi \in \mathcal{C}(X)$ and a valuation $\nu \in \mathbb{R}_{\geq 0}^X$, we denote by $\varphi(\nu) \in \mathbb{B}$ the truth value of φ obtained by substituting each occurrence of x in φ by $\nu(x)$. We thus define $[\![\varphi]\!] = \{\nu \in \mathbb{R}_{\geq 0}^X \mid \varphi(\nu) = \mathsf{tt}\}$.

Timed words and languages

Let Σ be a finite set called *alphabet*. Σ^* (resp. Σ^ω) is the set of finite (resp. infinite) sequences of elements of Σ and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We will call *finite words* on the alphabet Σ the elements of Σ^* and *infinite words* on the alphabet Σ the elements of Σ^ω.

We define $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ with $\varepsilon \notin \Sigma$, where ε represents a particular action (or letter), said to be *silent* or *non-observable*.

DEFINITION 1.1(TIMED WORD).– A *timed word* w on an alphabet Σ is a finite or infinite sequence $w = (a_0, d_0)(a_1, d_1) \cdots (a_n, d_n) \cdots$ such that for all $i \geq 0$, $a_i \in \Sigma$, $d_i \in \mathbb{R}_{\geq 0}$ and $d_{i+1} \geq d_i$.

A timed word $w = (a_0, d_0)(a_1, d_1) \cdots (a_n, d_n) \cdots$ on Σ can be considered as a pair $(a, d) \in \Sigma^\infty \times \mathbb{R}_{\geq 0}^\infty$ with $|a| = |d|$. The value d_i gives the absolute date (considering the system starts at date 0) of action a_i.

We denote $\mathit{Untimed}(w) = a_0 a_1 \cdots a_n \cdots$ the untimed part of w, and $\mathit{Duration}(w) = \sup_{d_i \in d} d_i$ the duration w.

We denote $\mathcal{TW}^*(\Sigma)$ (resp. $\mathcal{TW}^\omega(\Sigma)$) the set of finite (resp. infinite) timed words on Σ and we define $\mathcal{TW}^\infty(\Sigma) = \mathcal{TW}^*(\Sigma) \cup \mathcal{TW}^\omega(\Sigma)$.

DEFINITION 1.2(TIMED LANGUAGE).– A *timed language* L on Σ is a subset of $\mathcal{TW}^\infty(\Sigma)$.

Timed transition systems

We discuss systems that can be described by a labeled transition system, i.e. a set of states and a set of transitions between states, labeled by actions. In a given state, it is possible to change the state by executing the action by labeling one of the outgoing edges. A run in the transition system is then a sequence of transitions.

Timed transition systems (TTSs) are special transition systems for which two types of transitions are possible: action transitions and time transitions, modeling, discrete and continuous evolutions of the system, respectively.

DEFINITION 1.3(TIMED TRANSITION SYSTEM).– A (labeled) *timed transition system* is a 4-tuple $S = (Q, Q_0, \Sigma, \longrightarrow)$ where Q is a set of states, $Q_0 \subseteq Q$ is the subset of initial states, Σ is a finite set of actions (disjoint from $\mathbb{R}_{\geq 0}$), $\longrightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ is a set of edges called transition relation. When $(q, e, q') \in \longrightarrow$, we also write $q \xrightarrow{e} q'$. The transition relation consists of a *continuous* transition relation $\xrightarrow{d \in \mathbb{R}_{\geq 0}}$ and a *discrete* relation transition $\xrightarrow{a \in A}$.

We make the following standard assumptions on TTS:

– 0-DELAY : $q \xrightarrow{0} q'$ iff $q = q'$;

– ADDITIVITY: if $q \xrightarrow{d} q'$ and $q' \xrightarrow{d'} q''$ with $d,\ d' \in \mathbb{R}_{\geq 0}$, then $q \xrightarrow{d+d'} q''$;

– CONTINUITY: if $q \xrightarrow{d} q'$, then for all $d',\ d'' \in \mathbb{R}_{\geq 0}$ such that $d = d' + d''$, there exists q'' such that $q \xrightarrow{d'} q'' \xrightarrow{d''} q'$;

– TIME DETERMINISM: if $q \xrightarrow{d} q'$ and $q \xrightarrow{d} q''$ with $d \in \mathbb{R}_{\geq 0}$, then $q' = q''$.

DEFINITION 1.4(RUN IN A TTS).– A *run* ρ in a TTS S is a finite or infinite sequence of continuous and discrete transitions in S. The set of runs of a TTS S is denoted by $[\![S]\!]$.

A run can always be written as an alternating sequence of continuous (possibly by delay 0) and discrete transitions:

$$\rho = q_0 \xrightarrow{d_0} q_0' \xrightarrow{a_0} q_1 \xrightarrow{d_1} q_1' \xrightarrow{a_1} \cdots q_n \xrightarrow{d_n} q_n' \cdots$$

We note *Untimed*$(\rho) = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$ and *Duration*$(\rho) = \sum_{i=0}^{n} d_i$, if ρ is finite and of size n, and, when ρ is infinite, *Duration*$(\rho) = \sum_{i=0}^{\infty} d_i$, if the series converges (ρ is then said to be *Zeno*) and *Duration*$(\rho) = \infty$, otherwise.

DEFINITION 1.5(TRACE).– The *trace* of a run $\rho = q_0 \xrightarrow{d_0} q_0' \xrightarrow{a_0} q_1 \cdots q_k \xrightarrow{d_k} q_k' \cdots$ in a TTS is the timed word *trace*$(\rho) = (a_0, \delta_0) \cdots (a_k, \delta_k) \cdots$ such that $\delta_k = \sum_{i=0}^{k} d_i$.

A timed word w, possibly infinite, is *accepted* by the TTS $S = (Q, Q_0, \Sigma, \longrightarrow)$, if there exists a run of S starting from some initial state $q_0 \in Q_0$, and whose trace is w. We could also define a set of accepting (or repeated) states in which the run has to finish (or to visit infinitely often for infinite runs). Here, we consider that this set is Q. The *timed language* $\mathcal{L}(S)$ accepted by S is the set of timed words accepted by S. In the case of finite words, as all states are accepting any prefix of an accepted timed word is also accepted, and so, we say that the corresponding timed language is *prefix-closed*.

The *untimed language* $\mathcal{L}_u(S))$ of the TTS S is the set of words accepted by S in which time actions have been abstracted: $\mathcal{L}_u(S) = \{\mathit{trace}(\mathit{Untimed}(\rho)) | \rho \in [\![S]\!]\}$.

Finally, when a TTS is defined on Σ_ε, i.e. some discrete actions are labeled by ε, its language is defined as the language of the corresponding ε-abstract TTS.

DEFINITION 1.6(ε-ABSTRACT TTS).– Let $S = (Q, Q_0, \Sigma_\varepsilon, \longrightarrow)$ be a TTS. We define a corresponding *ε-abstract* TTS $S^\varepsilon = (Q, Q_0^\varepsilon, \Sigma, \longrightarrow_\varepsilon)$ by:

– $q \overset{d}{\longrightarrow}_\varepsilon q'$ with $d \geq \mathbb{R}_{\geq 0}$ iff there exists a run $\rho = q \rightarrow q'$ with $\mathit{trace}(\mathit{Untimed}(\rho)) \in \varepsilon^*$ and $\mathit{Duration}(\rho) = d$;

– $q \overset{a}{\longrightarrow}_\varepsilon q'$ with $a \in \Sigma$ iff there exists a run $\rho = q \rightarrow q'$ with $\mathit{trace}(\mathit{Untimed}(\rho)) \in \varepsilon^* a \varepsilon^*$ and $\mathit{Duration}(\rho) = 0$;

– $Q_0^\varepsilon = \{q \mid \exists q' \in Q_0 \mid \ \rho = q' \rightarrow q$ and $\mathit{Duration}(\rho) = 0 \wedge \mathit{trace}(\mathit{Untimed}(\rho)) \in \varepsilon^*\}$.

1.3. Timed models

Transition systems are very general and low-level formalism. A problem arises as their representations can be infinite, which greatly impairs our ability to perform automatic verification or control this formalism. Therefore, we will study higher level models, for which the representation is finite, and the behavior can be encoded by a (possibly infinite) transition system. This defines classes of transition systems for which the problems of interest are decidable or at least more easily expressible.

1.3.1. *Timed Automata*

Presentation

Timed Automata are basically non-deterministic finite automata equipped with real-valued clocks. They have been introduced by Alur and Dill [ALU 94] and extended with the notion of location invariant by Henzinger *et al.* [HEN 94]. The notion of time in TA relies both on the added clocks and on predicates involving these clocks.

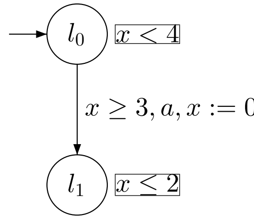

Figure 1.1. *A timed automaton*

These predicates can either be a *guard* associated with a transition and must be true for the transition to be taken, or an *invariant* associated with a location and must be true for the system to stay in that particular location.

Figure 1.1 describes a simple example of timed automaton. The initial location is l_0. The automaton has only one clock, named x, such that x is initially zero and the invariant of l_0 implies that the system can stay there during (strictly) less than four time units. As soon as three time units have elapsed, the guard of the transition between l_0 and l_1 is satisfied (and the invariant of l_1 will be satisfied if the transition is taken), so the transition can be taken. If we do so, the clock x is reset and l_1 becomes the active location.

Definitions

DEFINITION 1.7(TIMED AUTOMATON).– [HEN 94] A *Timed Automaton* is a 6-tuple $(L, l_0, X, \Sigma, E, Inv)$ where:

– L is a finite set of *locations*;

– $l_0 \in L$ is the *initial location*;

– X is a finite set of non-negative real-valued *clocks*;

– Σ is a finite set of *actions*;

– $E \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ is a finite set of *edges*. Let $e = (l, g, a, R, l') \in E$, then e is an edge between l and l', with the *guard* g, the action a and the set of clocks to reset R;

– $Inv \in \mathcal{C}(X)^L$ sets an *invariant* for each location.

We define the semantics of a timed automaton as a timed transition system.

DEFINITION 1.8(SEMANTICS OF A TA).– The semantics of a timed automaton $\mathcal{A}$ is a timed transition system $\mathcal{S}_\mathcal{A} = (Q, Q_0, \Sigma, \rightarrow)$ where:

– $Q = L \times (\mathbb{R}_{\geq 0})^X$;

– $Q_0 = (l_0, \mathbf{0})$;

– $\rightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ is the relation defined for $a \in \Sigma$ and $d \in \mathbb{R}_{\geq 0}$ by:
- the *discrete* transition relation: $(l, \nu) \xrightarrow{a} (l', \nu')$ iff $\exists (l, g, a, R, l') \in E$ s.t.

$$\begin{cases} g(\nu) = \mathsf{tt}, \\ \nu' = \nu[R \mapsto 0], \\ Inv(l')(\nu') = \mathsf{tt} \end{cases}$$

- the *continuous* transition relation: $(l, \nu) \xrightarrow{d} (l, \nu')$ iff

$$\begin{cases} \nu' = \nu + d, \\ \forall d' \in [0, d],\ Inv(l)(\nu + d') = \mathsf{tt}\ . \end{cases}$$

Decidability of classical problems

The *region graph* construction provides a universal tool for arguing about the decidability of several problems on TA [ALU 90, ALU 94]. Using this technique, it was shown, e.g. that reachability is decidable in PSPACE for TA [ALU 94] as well as for networks of TA [ACE 02], untimed language equivalence for TA is decidable in PSPACE [ALU 94], and untimed bisimilarity for TA is decidable in EXPTIME [LAR 97].

Practically more efficient algorithms are usually achieved by considering *zones* instead of regions (see e.g. [BEN 03]). Somewhat surprisingly even timed bisimilarity for TA is decidable. Using region graphs on a product construction, timed bisimilarity was shown to be decidable in EXPTIME [CER 93].

Unfortunately, timed language equivalence for TA is already undecidable [ALU 94]. In fact, even the universality problem (whether a given timed automaton generates *all* timed traces) is undecidable too.

1.3.2. *Time Petri nets*

Petri nets and time

The three main time extensions of Petri nets are *timed* Petri nets [RAM 74], *TPNs* [MER 74], and TAPNs [BOL 90, HAN 93].

In timed Petri nets, time is modeled by minimal durations (or exact when considering an "as soon as possible" execution of the net) for the firing of transitions. In TPNs, time is modeled by intervals constraining the possible firing dates of the transition. Moreover, time can be bound to timed transitions, timed places, timed arcs, or timed tokens. The resulting classes of TPNs are included in the corresponding classes of TPNs [PEZ 99].

Weak vs. strong semantics

We can further distinguish two semantics for TPN models: in weak semantics, timed actions are available in the span of the interval but can be ignored, and in strong semantics, time cannot progress if an action is possible and the value of the clock bound to it has reached the upper bound of time interval. The strong semantics permit to approach model urgency, which is an essential feature of real-time systems. Except for T-time Petri nets (where time intervals are bound to transitions) with strong semantics, the evolution of the net may also lead to the presence of so-called "dead" tokens in the net which cannot be used for any transition firing and may be sometimes difficult to interpret.

Single-server vs. multi-server semantics

We can finally distinguish the single-server semantics from multi-server semantics [BOY 01, BER 01]. The number of *clocks* to be considered is finite with the single-server semantics (one clock per transition, one per place, or one per arc), whereas in general it is not in the case of multi-server semantics.

Timed-arc Petri nets have been mainly studied with weak (lazy) multi-server semantics [HAN 93, RUI 99, FRU 00, NIE 01, RUI 00, ABD 01, ABD 07, SRB 05, BOU 08]: this means that the number of clocks is not finite but the firing of transitions may be delayed, even if this implies that some transitions are disabled because their input tokens become too old. In fact, in TAPNs, time is assigned to tokens and arcs are labeled by the intervals that restrict the age of available tokens. The reachability problem is undecidable [RUI 99] for this class of Petri nets but thanks to the weak semantics, it enjoys *monotonic* properties and falls into a class of models for which the coverability and boundedness problems are decidable [RUI 00, ABD 01, ABD 07].

Conversely, T-time Petri nets [MER 74, BER 91] and P-time Petri nets [KHA 96] have been studied only with a strong single-server semantics. They do not have the monotonic features of the weak semantics although the number of *clocks* is finite. The marking reachability and coverability problems are known to be undecidable [JON 77] but marking coverability, k-boundedness, state reachability, and liveness are decidable for *bounded* T-time Petri nets and P-time Petri nets with strong semantics.

Therefore, T-time Petri nets with a strong single-server semantics are most widely used for modeling real-time systems, and we will mainly focus on them in this book, when dealing with Petri nets. We will generally refer to them as simply TPN.

Timed-arc Petri nets are conceptually different from T-time Petri net model and since T-time Petri net and TAPNs are both the most widely used and studied for

single-server and multi-server semantics, respectively, they are discussed in detail in sections 1.3.2.1 and 1.3.2.2.

1.3.2.1. *T-time Petri nets*

T-time Petri nets [MER 74] are basically Petri nets equipped with time intervals $[\alpha(t), \beta(t)]$ bound to each transition t of the net. To be fired, a transition must be enabled without interruption for a duration between $\alpha(t)$ and $\beta(t)$.

1.3.2.1.1. Definitions

DEFINITION 1.9(T-TIME PETRI NET).– A T-time Petri net is a 7-tuple $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$, where:

– $P = \{p_1, p_2, \ldots, p_m\}$ is a finite non-empty set of *places*;

– $T = \{t_1, t_2, \ldots, t_n\}$ is a finite non-empty set of *transitions* $(T \cap P = \emptyset)$;

– ${}^\bullet(.) \in (\mathbb{N}^P)^T$ is the *backward incidence* function;

– $(.)^\bullet \in (\mathbb{N}^P)^T$ is the *forward incidence* function;

– $M_0 \in \mathbb{N}^P$ is the *initial marking* of the net;

– $\alpha \in (\mathbb{R}_{\geq 0})^T$ and $\beta \in (\mathbb{R}_{\geq 0} \cup \{\infty\})^T$ are functions giving for each transitions: the *earliest* and *latest* firing time ($\alpha \leq \beta$), respectively.

A marking M of the net is an element of $\mathbb{N}^P$ such that for any $p \in P$, $M(p)$ is the number of tokens in place p.

A transition t is said to be *enabled* by the marking M, if $M \geq {}^\bullet t$, i.e., in M, the number of tokens in each input place of t is greater or equal to the valuation of the arc between that place and t. We denote this by $t \in \mathsf{enabled}(M)$.

A transition t is said to be newly enabled by the firing of the transition t' from marking M, which we denote by $\uparrow\mathsf{enabled}(t, M, t')$, if t is enabled by the new marking $M - {}^\bullet t' + t'^\bullet$ but was not by the marking $M - {}^\bullet t'$. Formally speaking,

$$\uparrow\mathsf{enabled}(t, M, t') = ({}^\bullet t \leq M - {}^\bullet t' + t'^\bullet) \wedge ((t = t') \vee ({}^\bullet t > M - {}^\bullet t')).$$

Similarly, t is said to be *disabled* by the firing of t' from marking M, which we denote by $\mathsf{disabled}(t, M, t')$, if t is enabled by M but is not anymore by $M - {}^\bullet t' + t'^\bullet$.

By extension, we denote by $\uparrow\mathsf{enabled}(M, t')$ (resp. $\mathsf{disabled}(M, t')$) the set of transitions newly enabled (resp. disabled) by the firing of t' from M.

As for timed automata, we define the semantics of time Petri nets as a timed transition system.

DEFINITION 1.10(SEMANTICS OF A TPN).– The semantics of a T-time Petri net $\mathcal{N}$ is a TTS $\mathcal{S}_{\mathcal{N}} = (Q, q_0, \Sigma, \rightarrow)$ where:

– $Q = \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^T$;

– $q_0 = (M_0, \mathbf{0})$;

– $\Sigma = T$;

– $\rightarrow \subseteq Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ is the relation consisting of:

- the *continuous* relation transition defined for any $d \in \mathbb{R}_{\geq 0}$ by:

$$(M, \nu) \xrightarrow{d} (M, \nu') \text{ iff } \begin{cases} \nu' = \nu + d, \\ \forall t_k \in T, M \geq {}^\bullet t_k \Rightarrow \nu'(t_k) \leq \beta(t_k); \end{cases}$$

- the *discrete* relation transition defined for $t_i \in T$ by:

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \text{ iff } \begin{cases} M \geq {}^\bullet t_i, \\ \alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\ M' = M - {}^\bullet t_i + t_i^\bullet, \\ \forall t_k, \nu'(t_k) = \begin{cases} 0, \text{ if } \uparrow \mathsf{enabled}(t_k, M, t_i), \\ \nu(t_k), \text{ otherwise.} \end{cases} \end{cases}$$

When a discrete transition is possible from a state $s = (M, \nu)$ of (the semantics of) the net, we say that the corresponding transition of the net is *firable*.

DEFINITION 1.11(FIRABLE TRANSITION).– Let $s = (M, \nu)$ be a state of the semantics of a TPN. A transition t is said *firable* from s if $M \geq {}^\bullet t$ and $\alpha(t) \leq \nu(t) \leq \beta(t)$.

Note that, with this semantics, when a place contains more than necessary tokens to enable an outgoing transition, that transition is enabled only once. In particular, only one clock is implicitly associated with it, measuring the duration of the enabling. This is the *single-server semantics*. This semantics is illustrated in Figure 1.2, where we suppose that t_1 is firable. In Figure 1.2(a), t_1 and t_2 are enabled by marking M and by $M' = M - {}^\bullet t_1 + t_1^\bullet$ but not by $M - {}^\bullet t_1$. Hence, the transitions t_1 and t_2 are both *newly* enabled by the firing of t_1.

In Figure 1.2(b), t_1 and t_2 are enabled by marking M and by $M' = M - {}^\bullet t_1 + t_1^\bullet$, but also by $M - {}^\bullet t_1$. In this case, t_1 is *newly* enabled by its own firing t_1 (because it is the fired transition) but not t_2: t_2 *remains* enabled (and the associated clock keeps its value).

The semantics can be adapted to allow a transition to be enabled several times at once [BER 01]. We then consider the *multi-server semantics* and the number of clocks becomes potentially infinite.