Compositional Verification of Hybrid Systems using Simulation Relations

Goran Frehse

Copyright © 2005, Goran Frehse, Lindenberg im Allgäu
ISBN 90-9019824-5
IPA Dissertation Series 2005-14

Typeset with LaTeX 2ε
Printed by Print Partners Ipskamp, Enschede

The work in this thesis was carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). It was supported in part by the German Research Foundation (DFG) under grants KO 1430/6-1 and EN 152/29-1, the US Army Research Office (ARO) contract no. DAAD19-01-1-0485, the US National Science Foundation (NSF) contract no. CCR-0121547, and the Semiconductor Research Corporation under task ID 1028.001.

Compositional Verification of Hybrid Systems using Simulation Relations

een wetenschappelijke proeve op het gebied

van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor

aan de Radboud Universiteit Nijmegen

op gezag van de Rector Magnificus prof. dr. C.W.P.M. Blom,

volgens besluit van het College van Decanen

in het openbaar te verdedigen op

Maandag 10 Oktober 2005

des namiddags om 3:30 uur precies

door

Goran Fedja Frehse

geboren op 31 Januari 1973

te Lindenberg im Allgäu, Duitsland

Promotores:

Prof. dr. Frits W. Vaandrager
Prof. dr. Sebastian Engell, Universität Dortmund, Duitsland

Manuscriptcommissie:

Prof. dr. Thomas A. Henzinger, EPF de Lausanne, Zwitserland
Prof. dr. Kim Guldstrand Larsen, Aalborg Universitet, Denemarken
Prof. dr. Bruce H. Krogh, Carnegie Mellon University, VS

Preface

Many people contributed to this work, and it was very fortunate for me that the networking provided by national and international cooperative efforts, namely the German project KONDISK and the European projects VHS and AMETIST, provided vital connections and interaction with other researchers. I am deeply grateful to Prof. Frits W. Vaandrager and Prof. Sebastian Engell for supervising this thesis. Prof. Vaandrager has been of great help in giving this thesis structure, improving its content, tracking down errors and improving the readability. His clarity and precision of thought made the collaboration highly effective and a great pleasure. Prof. Engell has provided me not only with an interesting and challenging research topic, but also with the freedom and flexibility I needed in its pursuit. His guidance and analytical sense for the essential were as inspiring as they were indispensable. As a visiting researcher at Carnegie Mellon University, the numerous discussions with Prof. Bruce Krogh have substantially widened my horizon and reshaped my way of thinking. His enthusiasm and passion for science, his leadership and his instinct for meaning and purpose in research activities have been highly inspiring. Prof. Kim G. Larsen most generously invited me to several research visits to Aalborg University, during which he spent much time and spared no effort introducing me to several essential concepts of computer science, some of which this thesis builds on.

Special thanks go to the Deutsche Forschungsgemeinschaft (German Research Foundation), which financed a large part of my research under grants KO 1430/6-1 and EN 152/29-1, and to Prof. Willem-Paul de Roever, Univ.-Prof. Stefan Kowalewski and Prof. Yassine Lakhnech for initiating those projects together with Prof. Engell. My research at Carnegie Mellon was supported by the US Army Research Office (ARO) contract no. DAAD19-01-1-0485, the US National Science Foundation (NSF) contract no. CCR-0121547, and the Semiconductor Research Corporation under task ID 1028.001, and I am most grateful to Prof. Krogh, Prof. Rob A. Rutenbar, and the participating agencies for providing this funding. Part of my progress I owe to innumerable lecture notes, tutorials and presentations available on-line, some by the most renowned experts in the field. I enthusiastically applaud the contributors for sharing.

Throughout these years I have benefitted from the help of many friends and colleagues, and I would like to thank them for all their hard work, their valuable advice and critique, for lending me their ears, for all the pencils and markers we used up in long hours of discussions, for going out and having fun when a cerebral reset, soft

or hard, was called for, and for making research a wonderful and exciting adventure: Florian Pivit in Karlsruhe; Nanette Bauer, Stefan Krämer, Manuel Pereira Remelhe, Olaf Stursberg, Heinz Treseler and Michael Wieland in Dortmund; Ralf Huuck and Ben Lukoschus in Kiel; Emmanuel Fleury in Aalborg; and Zhi Han, Dong Jia, Jim Kapinski and Klaus Schmidt in Pittsburgh. I particularly thank Dr. Franz Franchetti and Prof. Markus Püschel from Carnegie Mellon University for their helpful advice and motivational support. Finally, a very special thanks goes to my wonderful parents Brigitte and Curt for their continuing support and encouragement.

Contents

1 Introduction **1**
1.1 Context 4
1.2 Overview 5

I Discrete Systems

Introduction **9**

2 Modeling with Labeled Transition Systems **11**
2.1 Preliminaries 11
2.2 Labeled Transition Systems 12
2.3 Modeling Directional Interaction 14
2.4 Modeling Shared Variables 17
2.5 Related Work 18

3 Simulation Relations for Labeled Transition Systems **21**
3.1 Simulation 22
3.2 Σ-Simulation 25
3.3 Computing Simulation Relations 33
3.4 Related Work 36

4 Assume-Guarantee Reasoning **39**
4.1 Non-circular A/G-Reasoning 40
4.2 Circular A/G-Reasoning with Simulation 41
4.3 Circular A/G-Reasoning with Σ-Simulation 54
4.4 Related Work 62

II Hybrid Systems with Discrete Interaction

Introduction **65**

5 Modeling with Hybrid Automata **67**
5.1 Hybrid Automata 67

5.2 Linear Hybrid Automata 71
5.3 Timed Transition System Semantics 75
5.4 Related Work 77

6 Simulation Relations for Hybrid Automata 81
6.1 Simulation Relations with Equivalence 81
6.2 Compositional Reasoning 85
6.3 Computing Simulation Relations in $\mathbb{R}^n$ 91
6.4 Related Work 96

7 Assume-Guarantee Reasoning 99
7.1 Non-circular Assume-Guarantee Reasoning 100
7.2 Circular Assume-Guarantee Reasoning 101
7.3 Related Work 105

III Hybrid Systems with Continuous Interaction

Introduction 109

8 Modeling with Hybrid I/O-Automata 111
8.1 Hybrid I/O-Automata 111
8.2 Hybrid Labeled Transition Systems 114
8.3 Related Work 117

9 Simulation Relations for Hybrid I/O-Automata 121
9.1 Trace- and TTS-Simulation 121
9.2 Compositional Reasoning 123
9.3 Related Work 132

10 Assume-Guarantee Reasoning 133
10.1 Non-circular Assume-Guarantee Reasoning 133
10.2 Circular Assume/Guarantee Reasoning 134
10.3 Related Work 138

11 PHAVer - A Novel Verification Tool for Hybrid Systems 139
11.1 Simulation Checking and Assume/Guarantee-Reasoning 141
11.2 Reachability Analysis 148
11.3 Managing Complexity 156
11.4 Related Work 164

Summary

12 Conclusions **167**
12.1 Discrete Systems 168
12.2 Hybrid Systems with Discrete Interaction 168
12.3 Hybrid Systems with Continuous Interaction 169
12.4 PHAVer 170
12.5 Future Research 171

Appendix

A PHAVer Language Overview **173**
A.1 General 173
A.2 Constants 173
A.3 Data Structures 174
A.4 Commands 175
A.5 Parameters 178

Bibliography **183**

Samenvatting **199**

Zusammenfassung **203**

List of Tables

3.1 Simulation relations for Ex. 3.1 24

4.1 Simulation relations for Ex. 4.4 49
4.2 Simulation relations for Ex. 4.5 50
4.3 Simulation relations for Ex. 4.6 after separate trimming 53

5.1 Side-by-side comparison of hybrid automata definitions 80

11.1 Analysis of extended tank level monitor model 149
11.2 Experimental results for the navigation benchmark 156

List of Figures

2.1 Graphical representation of a labeled transition system 13
2.2 Model of a boolean variable 18

3.1 Structure of compositional reasoning framework 22
3.2 Similar, yet not bisimilar, labeled transition systems 24
3.3 Justification of non-blocking requirement 26
3.4 Algorithm for checking simulation of LTS 34
3.5 Algorithm for checking Σ-simulation of LTS 35
3.6 Algorithm for checking simulation of LTS with a waiting list 35
3.7 On-the-fly algorithm for checking simulation of LTS 36
3.8 On-the-fly algorithm for simulation with deterministic specification . . 37

4.1 A/G-reasoning yields a false negative for Ex. 4.1 42
4.2 Algorithm for A/G-simulation by separate trimming 48
4.3 A/G-reasoning with separate trimming works for Ex. 4.4 48
4.4 Example with failed A/G-reasoning by separate trimming 50
4.5 Algorithm for A/G-simulation by composite trimming 51
4.6 A/G-reasoning fails in Ex. 4.6 because of the initial states 52
4.7 Algorithm for A/G-reasoning by separate trimming 57
4.8 Algorithm for A/G-reasoning by composite trimming 58
4.9 Reactor with two raw material tanks 59
4.10 LTS models for tank and controller 60
4.11 Global specification 60
4.12 Decomposed specifications 61

5.1 Tank with inlet valve and constant outflow 70
5.2 Tank model with level sensors 72
5.3 LHA-approximation of the tank model from Fig. 5.1(b) 74
5.4 Linear hybrid automaton with linear discrete time dynamics 74
5.5 Timed transition relation 76
5.6 Graph of references between hybrid systems definitions 79

6.1 Reactor with level monitoring controller 83
6.2 Level monitoring controller 83
6.3 Specification automata 84

6.4 Abstraction of level monitoring controller 90
6.5 Semi-algorithm for computing a simulation relation 92
6.6 Semi-algorithm for computing a bisimulation relation 92

8.1 Composition of TTSs with non-convex invariant 118

9.1 Tank level monitoring system 130
9.2 Specification of the composed system 130

10.1 Modular specifications for A/G-reasoning 135

11.1 Model of tank P_1 142
11.2 Model of controller P_2 143
11.3 Model of specification Q 143
11.4 Simulation relation for $P \preceq Q$ 145
11.5 A/G-specifications 146
11.6 Reachability Algorithm in PHAVer 150
11.7 Splitting a location along a hyperplane 153
11.8 Partitioning states with a set of candidate constraints 153
11.9 Reachable states of the navigation benchmark 155
11.10 Algorithm for limiting the number of bits of a constraint 157
11.11 Scheme of limiting the number of bits 158
11.12 Example for limiting the number of constraints 160
11.13 Reconstructing a polyhedron by angle prioritization 160
11.14 Reachable states of tunnel diode circuit 161
11.15 Reachable states of clocked tunnel diode circuit 162
11.16 Reduction in bits and constraints for the clocked tunnel diode circuit . 163

Chapter 1

Introduction

With the prevalent integration of microprocessors into almost every electrical device, appliance and technical process, software controllers of hardware systems play an increasingly important role. The interaction of software with physical devices and processes can exhibit complex, mixed continuous-discrete behavior that defies the analytical capabilities of classic feedback control theory, which is directed mainly at purely continuous-time and continuous-valued systems, as well as those of computer science, which deals with discrete-time and discrete-valued systems. Systems with mixed continuous-discrete behavior are called *hybrid systems*. They have been studied since the early 1990s and are the subject of a multitude of ongoing research activities. The proper functioning of hybrid systems is often highly desirable, as in the case of safety critical applications, or because the number of systems in place and high maintenance costs economically justify closer scrutiny. Many of such systems, e.g., embedded systems, must function correctly without human intervention, either because such an intervention is not possible at the time of operation, or because erroneous intervention would be disastrous. The latter was the case in an accident on July 1, 2002, when a Russian Tupolev TU154M passenger airplane found itself on a collision course with a Boeing B757-200 cargo plane, 35 000 feet above Lake Constance between Switzerland and Germany. A built-in safety system, the *Traffic Alert and Collision Avoidance System* (TCAS), correctly alerted the pilot to avoid the collision by pulling the plane up, while the human airspace controller in charge, impaired by adverse working conditions, ordered it to push down. The pilot followed the advice of the human controller, which resulted in a collision that killed 71 passengers and crew. An official investigation by the German Federal Bureau of Aircraft Accidents Investigation (BFU) included the following in its recommendations [SBB+04]:

> *ICAO [International Civil Aviation Organization] should change the international requirements [...] so that pilots flying are required to obey and follow TCAS resolution advisories (RAs), regardless of whether contrary ATC [Air Traffic Control] instruction is given prior to, during, or after the RAs are issued. Unless the situation is too dangerous to comply,*

> *the pilot flying should comply with the RA until TCAS indicates the airplane is clear of the conflict. [...] ICAO should ensure a high level of acceptance and confidence of pilots staff in ACAS by improving education and training.*

These conclusions underline the importance of designing software for critical applications with an utmost level of confidence, as well as the need to educate about its capabilities and limitations. It is the goal of *formal verification*, i.e., a mathematical modeling of the system and formal proof of its properties, to ensure the safety of a system to such a high level of confidence, and critical parts of the TCAS have been formally verified [LLL00]. Because of the inherent complexity, the analysis is limited to critical components of the system at the design level and usually requires a certain level of abstraction from physical details. Other causes for a malfunction of the system, such as hardware failures, are better addressed using methods such as hazard analysis, or statistical reliability analysis.

The necessity of high-confidence systems is expected to increase over the next years as new technologies spread software controllers into safety critical consumer applications. E.g., in the automotive sector drive-by-wire, pre-safe systems and other driver assisting technologies are destined to proliferate within the next decade [All03]. In health care, critical applications include automatic heart defibrillators, health monitoring systems and automatic medication devices. A significant portion of high-confidence systems will be part of homeland security and anti-terrorism systems such as soft walls, which are safety controllers designed to prevent planes from flying into designated areas [Cat04].

While formal verification has become a standard tool in the design of discrete systems such as digital circuits, its application in hybrid systems in hindered by the inherent complexity of the discrete-continuous behavior. Most systems are naturally divided into interacting components, also referred to as modules or subsystems, based on their physical manifestation or function. The behavior of the system is then a subset of the product of the behaviors of the components, with a corresponding increase in complexity. Verification is limited in its application to relatively simple systems because the computational cost increases exponentially with the number of components and variables involved. This phenomenon is referred to as the *state explosion problem*. While the state explosion also takes place in purely discrete systems, hybrid systems are in addition subjected to the complexities of higher dimensional continuous dynamics, which increase dramatically with the number of state variables.

To enable the verification of large hybrid systems, we follow two fundamental approaches that are successfully being applied in the discrete domain: abstraction and compositional reasoning. In an *abstraction* of the system, information is discarded that is irrelevant to the proof of the properties. Such a simplified model must be conservative with respect to those properties. A *compositional analysis* of the system examines parts of the system in such a way that properties of the entire systems can be deduced. A particularly effective form of compositional proof is assume/guarantee-reasoning, in

which the specification of a subsystem is used as an assumption restricting the behavior of the other subsystems, thus leading to a circular or chain-rule type of proof.

To formally show that a model is indeed an abstraction of another we construct a *simulation relation* between the states of the models. A state *simulates* another if the same, or more, behavior is possible. Intuitively, this is conservative in the sense that if something cannot happen in the abstraction, it will not happen in the original either. The same approach can be used to compare a system and its specification, so we equivalently refer to the refined model as the system, and to the abstract model as the specification. Whether a compositional analysis is always valid depends on how simulation is defined, and not all types of simulation are computable for hybrid systems. It is the central goal of this thesis to show compositionality for a type of simulation that is computable, and in its pursuit we accomplished advances in the following areas:

- The classic concept of simulation relations requires the system and the specification to range over the same set of actions. We propose an extension to arbitrary actions, and show that it is an upper bound on any such extension that preserves compositionality. This extension simplifies models and proofs, and is relevant to both discrete and hybrid systems.

- We propose a novel circular assume/guarantee rule for simulation relations applicable to discrete as well as hybrid systems. An assume/guarantee rule is a type of compositional proof that permits tighter specifications by combining the modules of the system with assumptions about the behavior of their environment, and thus yields an overall smaller set of problems. If these assumptions depend on the specifications in a circular manner, the rule is only sound if additional conditions are fulfilled that break this circularity. We establish assume/guarantee conditions for simulation relations and show that they are sufficient and necessary for the existence of a simulation relation.

- We show that simulation for hybrid systems is compositional if the subsystems share no variables. We illustrate how for such cases simulation relations can be computed algorithmically and applied to minimize models.

- For hybrid systems with shared variables we present a formalism that is compositional for two important classes of hybrid systems: hybrid systems with unrestricted inputs and hybrid systems defined by linear predicates, convex invariants and piecewise constant derivatives, so-called *linear hybrid automata* (LHA). Because any hybrid system can be approximated arbitrarily close by a LHA, this makes any hybrid system accessible to compositional reasoning by using such an overapproximation.

- We implement algorithms for checking simulation and assume/guarantee reasoning in PHAVer, a verification tool for linear hybrid automata that is publicly available [Fre05].

1.1 Context

Simulation relations, introduced by Milner [Mil71], are readily applied in a compositional analysis [GL91], an approach that only recently has seen an increased interest in the area of hybrid systems [HQRT02, TP02, TPL02]. Both the system and its specification are given in the form of a hybrid automaton, i.e., a state-transition system in which the dynamics of the continuous variables depend on which node, or *location*, the system is in. Hybrid automata can be interpreted semantically as infinite labeled transition systems (LTSs) with a structure imposed by the separation between continuously changing variables and discrete changes in their dynamics. This interpretation is called a *timed transition system*. Simulation for hybrid systems can be defined based on these TTS-semantics, as proposed by Henzinger in [Hen96], and preserve safety properties. Continuous variables in the system and the specification are related by an equivalence relation that is imposed on the simulation relation as an upper bound. So far not further explored in literature, this approach turns out to be cumbersome in practise, and the structural separation between locations and variables is lost. Lynch et al. use in their framework of Hybrid Input/Output Automata a definition of simulation based on trajectories [LSV03]. While this approach is inherently compositional, it does not easily admit a finitary representation and therefore is not immediately open to practical applications.

A framework for compositional reasoning based on simulation relations for discrete systems was proposed by Grumberg and Long [GL91]. We adopt this framework, and extend it with our notion of simulation over arbitrary alphabets. A special form of compositional reasoning is assume/guarantee-reasoning (A/G-reasoning). It combines the system and its specification during the analysis in two distinct forms, circular and non-circular. In a chain argument over the entire system, referred to as *non-circular assume-guarantee reasoning*, each subsystem is conform with the specification under the assumptions that are already established as true, and guarantees in turn a behavior that is used as an assumption for the analysis of the next subsystem. The core of *circular assume-guarantee* is to allow a circular dependency between the assumptions and the guarantees. If such a circularity exists, addition reasoning is required to guarantee soundness of the conclusion.

A/G-reasoning was pioneered by Jones [Jon81], and Misra and Chandy [MC81]. While the soundness of non-circular A/G-reasoning directly follows from compositionality, the circular version requires further scrutiny of the system. Henzinger et al. showed in [HQRT02] that A/G-reasoning is sound for receptive discrete systems. Extensions of A/G-reasoning to hybrid systems proposed in literature have not been based on simulation, but also depend on receptiveness, see [AH97, HQR98, HMP01].

So far there exists no tool support for either checking simulation or assume/guarantee-reasoning of hybrid systems. The tool most related to PHAVer is HyTech [HHWT97], a tool for checking reachability of linear hybrid automata. While powerful in its functionality, its infinite precision arithmetic is limited to a small number of digits, and therefore quickly leads to overflow problems that severely limit its ap-

plicability. CheckMate [CK03], a prominent verification tool for non-linear hybrid automata based on Matlab/Simulink supports non-determinism only in the set of initial states. Its floating point arithmetic is, strictly speaking, not sound, and does not support open sets, which can lead to problems, e.g., when modeling if–else structures. The tool d/dt performs a reachability analysis of piecewise linear and nonlinear hybrid systems by overapproximating sets of states with a rectangularized representation. It is algorithmically sound but restricted by a termination condition that depends on an under-approximation of the reachable states, and also prone to the flaws of finite precision arithmetic.

1.2 Overview

This thesis is divided into three parts: In Part I we present simulation and compositional reasoning for labeled transition systems, thus providing the foundation for the subsequent extension to hybrid systems. In Part II these results are extend in a straightforward manner to hybrid systems whose components do not share continuous variables. In Part III, we use a modified semantics called *hybrid labeled transition systems* that enable us to handle hybrid systems with shared variables. We also present our verification tool PHAVER. In each of the parts, we define simulation and present compositional and assume/guarantee reasoning for the respective class of systems. An overview on related work can be found at the end of each chapter. We proceed with an outline of each of the parts.

We begin in Chapter 2 with recalling labeled transition systems, and discuss how they can be used to model systems with variables and their directed interaction in a sender/receiver fashion. In Chapter 3, the classic concept of simulation relations, which only compares LTS of the same alphabet, is extended to Σ-*simulation*, which covers arbitrary alphabets. We use chaos automata to obtain basic theorems of compositional reasoning and relate classic simulation with Σ-simulation. We present compositional reasoning in three major rules: compositionality, decomposition of the specification, which in this form is novel as far as we know, and non-circular assume/guarantee-reasoning. In Chapter 4 we present our novel proof rule for circular assume/guarantee reasoning, and show two ways to implement such a proof in practise: by trimming bad states separately from each simulation relation, and by trimming bad states based on a composite list of potentially bad states. The presentation of the rules is divided up for simulation and Σ-simulation since they take on slightly different forms, with Σ-simulation being the more compact one.

In Part II we extend the compositional framework of Part I to hybrid systems that do not share any variables. In Chapter 5 we introduce the basics of hybrid automata and their labeled transition system semantics. We give a summary of well-established efficiency improvements of the reachability analysis, since they are equally relevant to computing simulation relations and the set of reachable states can be used to initialize simulation relations before the fixed-point computation. In Chapter 6 we define simu-

lation for hybrid systems based on their TTS-semantics. Equivalence relations are used to describe the relation of variables in the system and the specification, e.g., to express which variables refer to the same physical property and which are unrelated. We show that this definition is compositional as long as the hybrid automata do not share any variables. The compositional framework of Part I is applied to hybrid systems, and we provide criteria for the equivalence relations under which the compositional rules hold. We describe how simulation relations can be computed in $\mathbb{R}^n$ and summarize methods to improve the efficiency. The extension of the assume/guarantee-reasoning, given in Chapter 7, is straightforward under the premise that there are no shared variables, and only differs with those from Part I by giving upper bounds on the equivalence relations that are admissible.

In order to deal with shared variables, Part III introduces a slightly different formalism for hybrid systems. In Chapter 8 we present a class of hybrid input/output automata (HIOA), which are hybrid automata with dedicated input- and controlled variables, of which only a subset of output variables is available to other automata. We propose an extended version of labeled transition systems, which inherit the separation of variable valuation and locations from hybrid automata. They have an environment action that receives special treatment in parallel composition, which allows us the definition of TTS-semantics of HIOA that preserve the invariants. Chapter 9 defines simulation for HIOA, and discusses the differences between simulation based on trajectories and simulation based on TTS-semantics. The equivalence between variables in the system and the specification is imposed by demanding that input and output variables are considered identical in both, while state variables that are not outputs are not related. Compositional reasoning is shown to be possible by applying the TTS-semantics to the subsystems, and, essentially, analyzing the behavior of the composed TTSs. For general case this amounts to abstracting from the continuous interaction, while discrete changes of the variables are taken into account. We show the that there is no overapproximation, and therefore full compositionality, when hybrid systems have either unrestricted inputs, or convex invariants and piecewise constant bounds on the derivatives, i.e., LHA. In Chapter 10, we adapt our assume-guarantee-rule to HIOA. However, we suffer the same abstraction of the continuous interaction as in the rest of the compositional framework, so that the rule is mainly useful for hybrid systems with unrestricted inputs and LHA. In the final chapter of Part III, we present our tool PHAVer, which we used successfully to verify reachability problems of hybrid systems with linear and affine dynamics. We illustrate the algorithms for on-the-fly approximation of affine dynamics, and for managing complexity by limiting the number of bits and constraints in polyhedral computations. We also implemented the algorithms for checking simulation and assume/guarantee reasoning presented in this thesis, and illustrate them with some examples.

Conclusions from each of the parts are collected in Chapter 12. The appendix contains an overview of the input language to PHAVer, a summary of this thesis in Dutch, a summary in German, and a curriculum vitae.

Part I

Discrete Systems

Introduction

In Part I we revise and extend a framework for compositional reasoning to labeled transition systems. We will build on this framework in the subsequent parts, where it is applied to hybrid systems. The merit of treating discrete systems before moving on to hybrid systems lies in the clarity and conciseness of definitions and proofs, the hybrid extension of which can be cumbersome at times. We have tried to keep definitions and proofs consistent in their structure throughout the thesis, and hope this part may serve as an introduction as well as a reference for the hybrid context.

One of the most fundamental concepts for modelling systems is the state-transition paradigm. In the context of systems theory, a *state* is a configuration of the system such that, given all future inputs, the entire future of the system is determined. In the presence of non-determinism, the state provides knowledge about the set of the possible future behavior. The *behavior* of a system is reflected in changes of its state called *transitions*. A transition is a directed edge that connects a state to another if the system can change to that state. The combination of states and transitions is a directed graph called a *state-transition system*. A state change can be associated with a qualifier, e.g., time elapse. To capture this, a label is attributed to each transition, which leads to the general model paradigm of *labeled transition systems*.

In the context of modeling and analysis, labeled transition systems fall into two main categories: *finite* and *infinite* systems, depending on whether the set of states, transitions or labels is finite or not. This distinction is made since algorithms based on explicit enumeration of states terminate for finite systems, while infinite systems require symbolic algorithms, i.e., algorithms that operate on sets of states. From this point of view, a hybrid system is simply an infinite labeled transition system with a special structure: by introducing real-valued variables the infinitely many states and transitions can be represented by a finite labeled graph, in which the nodes and arcs describe entire sets of states and transitions. This is somewhat analogous to the way a differential equation is a finite representation of a possibly infinite set of functions. With such an interpretation, many properties and results for labeled transition systems can be applied to hybrid automata, which is why we introduce the central concepts for labeled transition systems before proceeding with their extension and adaptation to hybrid systems in Parts II and III.

Most systems are naturally subdivided into components that are distinct in their physical or functional manifestations, in the following called *modules* or *subsystems*.

A divide-and-conquer approach models those parts and combines the separate analyses of the parts to provide some implication about the properties of the entire system. This is referred to as *compositional reasoning*. The interaction of modules can be captured in with discrete events, or transitions, and with variables that are shared between systems, and which can enable or disable transitions. In practical applications, this interaction is often considered to be directional, i.e., some modules govern certain behavior, while others merely react to it. One way to deal with directional interaction and shared variables is to define a powerful, but consequently complex modeling formalism that explicitly takes those phenomena into account, e.g., I/O automata [LT87, Lyn96] or C/E systems [SK91]. In contrast, in Parts I and II we use a simple formalism and discuss how directional interaction and shared variables can be modeled consistently within this framework. While this limits the expressiveness of our models, it allows us to apply compositional reasoning in a simple and straightforward manner, as will be shown in Chapter 3. The interaction of shared continuous variables limits the ways in which a compositional analysis can be performed. Such a formalism will be the subject of Part III. Our conclusions for Part I can be found in Sect. 12.1, pp. 168.

Chapter 2

Modeling with Labeled Transition Systems

In the following section we summarize some basic mathematical concepts and notation. We introduce labeled transition systems, their semantics and their interaction through parallel composition in Sect. 2.2. The modeling of directional interaction with is synchronization labels discussed in Sect. 2.3, and Sect. 2.4 describes models of shared variables with dedicated read- and write-access. Section 2.5 summarizes related work for this chapter.

2.1 Preliminaries

The following definitions summarize some fundamental mathematical conventions on relations, which will be extensively used in throughout the subsequent chapters.

Definition 2.1 (Binary Relation)**.** *A binary relation over sets X and Y is an ordered triple $R = (X, Y, G(R))$, with a* **graph** *$G(R) \subseteq X \times Y$ of the relation R. Usually, R is identified with $G(R)$. If elements $x \in X$ and $y \in Y$ are in the relation, this is denoted as $(x, y) \in R$, $R(x, y)$ or xRy. A binary relation over $X \times X$ is simply called a relation over X. It can be:*

- *reflexive: xRx*
- *symmetric: $xRz \Rightarrow zRx$*
- *antisymmetric: $xRz \wedge zRx \Rightarrow x = z$*
- *transitive: $xRy \wedge yRz \Rightarrow xRz$.*

To shorten a frequently used notation for a given set $X' \subseteq X$, we write $R(X')$ for the set $R(X') = \{y | \exists x \in X' : (x, y) \in R\}$.

Definition 2.2 (Preorder)**.** *A* **preorder** *is a relation that is reflexive and transitive.*

Definition 2.3 (Precongruence)**.** *A binary relation $\preceq$ is a* **precongruence** *with respect to an n-ary operator $f(x_1,\ldots,x_n)$ if it is invariant under f:*

$$xRy \Rightarrow f(x,x_2,\ldots,x_n)Rf(y,x_2,\ldots,x_n)$$

Definition 2.4 (Partial Order)**.** *A* **partial order** *is a relation that is reflexive, antisymmetric and transitive.*

A partial order can be constructed from a preorder $\preceq$ by defining an equivalence relation $\approx$ over $X \times X$ such that $x \approx y \Leftrightarrow (x \preceq y \wedge y \preceq x)$. The relation implied by $\preceq$ over the quotient set $X/\approx$ (the set of all equivalence classes defined by $\approx$) then forms a partial order.

Definition 2.5 (Equivalence Relation)**.** *An* **equivalence relation** *is a relation that is reflexive, symmetric and transitive.*

2.2 Labeled Transition Systems

A labeled transition system is a directed graph with labeled edges. In addition to states and transitions, a set of labels called *alphabet* is associated with the system. All labels on transitions must be from that alphabet. The following definition also includes a dedicated set of initial states, from which all behavior of the system originates:

Definition 2.6 (Labeled Transition System)**.** [Kel76] *A labeled transition system (LTS) is a quadruple $P = (S_P, \Sigma_P, \rightarrow, P_0)$ with*

- *a set S_P of* **states**,
- *a set of synchronization labels Σ_P called* **alphabet**,
- *a* **transition relation** $\rightarrow \subseteq S_P \times \Sigma_P \times S_P$ *and*
- *a set of* **initial states** $P_0 \subseteq S_P$.[1]

The semantics of a labeled transition system is captured by the concept of a *run*. A run is considered a behavior of the system only if it begins in one of the initial states, in which case it is called an *execution* of the system:

Definition 2.7 (Run, Execution,Reachability)**.** *A* **run** *of a labeled transition system P is a finite or infinite sequence of states and labels*

$$\sigma = p_0 \xrightarrow{\alpha_0} p_1 \xrightarrow{\alpha_1} p_2 \xrightarrow{\alpha_2} \ldots$$

such that for each pair of states $p_i, p_{i+1}, i \geq 0$ there exists a transition $p_i \xrightarrow{\alpha_i} p_{i+1}$. It is an **execution** *of P if $p_0 \in P_0$. A state p is called* **reachable** *if there exists an execution with $p = p_i$ for some $i \geq 0$.*

[1] We allow an empty set of initial states to be consistent with hybrid automata. We use labeled transition systems to define the semantics of hybrid automata. Since hybrid automata can have an empty set of initial states, see Def. 5.2, labeled transition systems should, too.

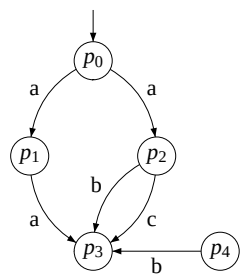


Figure 2.1: Graphical representation of a labeled transition system

Reachability can be used to specify many safety properties of interest, and is implemented in most verification tools for hybrid systems.

Example 2.1. *Figure 2.1 shows a graphical representation of a LTS P, where*

- $S_P = \{p_0, p_1, p_2, p_3, p_4\}$,
- $\Sigma_P = \{a, b, c\}$,
- $\rightarrow = \{(p_0, a, p_1), (p_0, a, p_2), (p_1, a, p_3), (p_2, b, p_3), (p_2, c, p_3), (p_4, b, p_3)\}$, *and*
- $P_0 = \{p_0\}$.

The states are represented by edges, and the transitions by vertices in the graph. The initial state is marked with an arrow. The executions of P are

$$p_0 \xrightarrow{a} p_1 \xrightarrow{a} p_3,$$
$$p_0 \xrightarrow{a} p_2 \xrightarrow{b} p_3, \text{ and}$$
$$p_0 \xrightarrow{a} p_2 \xrightarrow{c} p_3.$$

The states p_0, p_1, p_2, and p_3 are reachable, while p_4 is not.

The analysis of a system is greatly simplified if its behavior is *deterministic*, i.e., uniquely defined. We differentiate between the deterministic future of a single state, and call it successor-determinism, and the deterministic future of the automaton, for which we require that there be only one initial state.[2]

Definition 2.8 (Determinism)**.** *A labeled transition system P is* **successor-deterministic** *if in each state the label determines uniquely the successor state, i.e., for all $\alpha \in \Sigma$, $p, p'_1, p'_2 \in \Sigma_P$ holds:*

$$(p \xrightarrow{\alpha} p'_1 \wedge p \xrightarrow{\alpha} p'_2) \Rightarrow (p'_1 = p'_2)$$

It is **deterministic** *if it is successor-deterministic and has only one initial state.*

[2]One could generalize to the case where there is a set of initial states, but all outgoing transitions lead to the same target state, which in turn would guarantee a deterministic future. However, this seems to have no advantage.

Labeled transition systems interact by synchronizing on transitions with a common label. Transitions with labels that are not in the alphabet of the other system take place independently. The *parallel composition* operator merges two labeled transition systems to a single new one, whose states consist of a pair of states of the original systems.

Definition 2.9 (Parallel Composition)**.** [Kel76] *Consider labeled transition systems* $P = (S_P, \Sigma_P, \rightarrow_P, P_0)$ *and* $Q = (S_Q, \Sigma_Q, \rightarrow_Q, Q_0)$*. The* **parallel composition** $P||Q$ *is a labeled transition system* $C = (S_C, \Sigma_C, \rightarrow_C, C_0)$ *where* $S_C = S_P \times S_Q$*,* $\Sigma_C = \Sigma_P \cup \Sigma_Q$*,* $C_0 = P_0 \times Q_0$ *and the transition relation is given by the rules:*

$$(i)\ \frac{\alpha \in \Sigma_P \cap \Sigma_Q \wedge p \xrightarrow{\alpha}_P p' \wedge q \xrightarrow{\alpha}_Q q'}{(p,q) \xrightarrow{\alpha}_C (p',q')}$$

$$(ii)\ \frac{\alpha \in \Sigma_P \backslash \Sigma_Q \wedge p \xrightarrow{\alpha}_P p'}{(p,q) \xrightarrow{\alpha}_C (p',q)} \qquad (iii)\ \frac{\alpha \in \Sigma_Q \backslash \Sigma_P \wedge q \xrightarrow{\alpha}_Q q'}{(p,q) \xrightarrow{\alpha}_C (p,q')}$$

A labeled transition system can be represented graphically as a labeled graph. The following example includes figures in which the states are represented by rounded boxes, labeled with a name, and the transitions are arrows between the boxes, annotated with the transition label. In cases where drawing a transition arrow to its target state is impractical, we simply write the name of the state location at the tip of the arrow. Initial states are marked by a small incoming arrow with no source state.

2.3 Modeling Directional Interaction

The simultaneous triggering of state changes in several components takes place by synchronizing transitions that have a common label. The next two sections deal with basic aspects of such models and presents ways to model directional interaction and shared variables in a manner that will allow the use of the compositional methods in the subsequent chapters. While some modeling requirements seem complicated and work-intensive, e.g., introducing a new label for each sender and copying transitions, they are easily implemented as part of a graphical editor or analysis tool and in practise do not overly increase the complexity of the analysis. [3]

When modelling physical systems, a causal relationship is usually associated with signals. In the strict sense, causality is associated with a precedence in time. But more importantly, while some parts of the system act independently, others wait and act according to signals. For example, a controller signals a valve to open and proceeds

[3] Such an increase was witnessed in the translation from directional interaction in C/E systems to the synchronization labels of linear hybrid automata [Kow96, KEPS99]. However, that translation involved adding states as well as transitions, while the translation proposed in Sect. 2.3 only adds transitions. Experimental results indicate that the analysis is more sensitive to the number of states.

with its operation whether the valve opens or not. In the abstract setting of automata, this is captured by the notion that the sender of a signal behaves independently of the reception. A receiver can wait for the signal and change its state accordingly, or simply do nothing. The deciding factor is that the receiver cannot block the sender in its transition. We define non-blocking as follows:

Definition 2.10 (Non-Blocking)**.** *A labeled transition system P with alphabet Σ_P is* **non-blocking** [4] *on a label α if either $\alpha \notin \Sigma_P$ or for all states p there exists an outgoing transition with label α:*

$$\forall p : \exists p' : p \xrightarrow{\alpha} p'. \tag{2.1}$$

Consider a system consisting of multiple automata. With Def. 2.10, an automaton is a ***receiver*** of a label α if it is non-blocking on α. Conversely, it is a ***sender*** of α if all other automata with α in their alphabet are receivers of α. Note that a an automaton can be a receiver as well as a sender, or neither.

While being a receiver is a local property of an automaton, it depends on the entire system whether an automaton is a sender of a label. Any automaton of the system could block the label and therefore lead the sender to a dead-lock. This can quite easily lead to modelling errors. They can be avoided by introducing dedicated semantics for sending and receiving labels. A consistency check can then verify that all automata observe the sending and receiving directives. An alternative consists of automatically forcing non-blocking for all receivers by augmenting their transition relations:

Definition 2.11 (Forced Non-Blocking)**.** *The* **forced non-blocking** *transition relation of a labeled transition system P is given by*

$$\rightarrow' = \rightarrow \cup \{ (p, \alpha, p) \mid \nexists p' : p \xrightarrow{\alpha} p' \} \tag{2.2}$$

Forced non-blocking can easily be implemented by a preprocessing stage in analysis tools. On top of helping to avoid modeling errors, it relieves models of cluttering self-loops and can greatly enhance their readability. The following propositions describe how non-blocking behaves under composition:

Proposition 2.12. *If Q is non-blocking on $\Sigma \subseteq \Sigma_P$, then a transition with label Σ takes place in $P||Q$ if and only if it takes place in P, i.e., for all $\alpha \in \Sigma, (p,q) \in S_P \times S_Q, p' \in S_P$:*

$$(\exists q' : (p,q) \xrightarrow{\alpha} (p',q')) \Leftrightarrow p \xrightarrow{\alpha} p'. \tag{2.3}$$

Proof. (Sufficient) Assume that there exists a q' such that the transition $(p,q) \xrightarrow{\alpha} (p',q')$ exists. Since $\alpha \in \Sigma_P$ it holds with Def. 2.9 of parallel composition, that either (i) $\alpha \notin \Sigma_Q$ and $p \xrightarrow{\alpha} p'$, or (ii) $\alpha \in \Sigma_Q$ and $p \xrightarrow{\alpha} p'$ as well as $q \xrightarrow{\alpha} q'$, which proves the

[4] In an input/output setting, non-blocking is also referred to as input-enabledness.

sufficient implication. (Necessary) Assume that there is a transition $p \xrightarrow{\alpha} p'$. Since Q is non-blocking in α, Def. 2.10 yields that there exists a q' with $q \xrightarrow{\alpha} q'$. With Def. 2.9 of parallel composition, this implies a transition $(p,q) \xrightarrow{\alpha} (p',q')$, which concludes the proof. □

Proposition 2.13. *For labeled transition systems P and Q, $P||Q$ is non-blocking on α if and only if P and Q are non-blocking for α.*

Proof. There are four cases to be considered:

(i) $\alpha \notin \Sigma_P \cup \Sigma_Q$: By Def. 2.9 of parallel composition, $\alpha \notin \Sigma_{P||Q}$, and trivially $P||Q$, P and Q are non-blocking for α.

(ii) $\alpha \in \Sigma_P \backslash \Sigma_Q$: By definition, Q is non-blocking on α. Assume that $P||Q$ is non-blocking on α, so that for any (p,q) there exist (p',q') with a transition $(p,q) \xrightarrow{\alpha} (p',q')$. By Def. 2.9 of parallel composition, $\alpha \notin \Sigma_Q$ implies a transition $p \xrightarrow{\alpha} p'$ in P. Since this is the case for all p, P is non-blocking, which proves the sufficient implication.

Assume that P is non-blocking on α. Then for any p there exists p' and a transition $p \xrightarrow{\alpha} p'$, and with $\alpha \notin \Sigma_Q$, Def. 2.9 of parallel composition implies a transition $(p,q) \xrightarrow{\alpha} (p',q)$. Therefore $P||Q$ is non-blocking on α.

(iii) $\alpha \in \Sigma_Q \backslash \Sigma_P$: The argument is symmetric to (ii).

(iv) Otherwise $\alpha \in \Sigma_P \cap \Sigma_Q$: Assume that $P||Q$ is non-blocking on α, so that for any (p,q) there exist (p',q') with a transition $(p,q) \xrightarrow{\alpha} (p',q')$. Definition 2.9 of parallel composition implies transitions $p \xrightarrow{\alpha} p'$, $q \xrightarrow{\alpha} q'$ in P and Q. Since this is the case for all p and q, both P and Q are non-blocking, which proves the sufficient implication.

Assume that P and Q are non-blocking on α. Then for any (p,q) there exist (p',q') and transitions $p \xrightarrow{\alpha} p'$, $q \xrightarrow{\alpha} q'$. Definition 2.9 of parallel composition implies a transition $(p,q) \xrightarrow{\alpha} (p',q)$, so that $P||Q$ is non-blocking on α.

□

It follows immediately from Prop. 2.13 that blocking is invariant under composition. We do not restrict the requirement of non-blocking to states that are reachable from the initial state because instead the model can easily be adapted accordingly.

In cases where multiple automata P_i are senders of a label α, the only solution is to rename the label α to α_i in P_i and in the receivers insert for each transition with label α a copy that has the label α_i. Multiple receivers do not interfere with each other. According to Prop. 2.13, their composition is still a receiver, and with Prop. 2.12 a transition in the composition corresponds to a transition in both of the receivers.

2.4 Modeling Shared Variables

Many technical systems include quantities, switches or other memory of some form that could be modeled quite conveniently using variables. Each value of a variable corresponds to a state and the formalism of labeled transition systems would be easily extended to include variables in the form of a state vector or otherwise. However, shared variables in hybrid automata require a special treatment in compositional analysis, as will be discussed in Part III. For the formalism presented in this thesis, it is therefore advantageous to model such shared variables with synchronization labels where possible.

Every variable that takes only finitely many values can be modelled using synchronization labels. The difference between shared variables and a model using solely synchronization labels lies in the fact that the labeled model includes an explicit representation of the possible changes in the variable. With synchronization labels, automata can only block transitions, i.e., prevent a variable from taking values that it would be able to take without the interference. Therefore the set of values of a variable under the influence of an automaton is a subset of the values it would take without it, so that safety properties are preserved. This is formally captured by compositionality as it will be defined in Sect. 3.1. On the other hand, in a model with shared variables an automaton could set the variable to a new value that otherwise might never occur, and thus change safety properties.

In order to properly model read- and write-access of variables, a sender/receiver semantics as introduced in Sect. 2.3 should be applied. Write-access corresponds to a sender semantics, while the automaton that models the variable is the receiver. In other words, each automaton accessing the variable should have its own set of labels for that purpose, and the variable model should be non-blocking on those labels. Read-access is characterized by two conditions: The variable should not change its state through read-access, and in each state of the variable there should be at least one transition with a read-access label. That way an accessing automaton is not deadlocked if it offers all possible read-access labels. Formally, the model of a shared variable can be described as follows:

Definition 2.14 (Shared Variable Model)**.** *A* **shared variable model** *in a labeled transition system P is defined by $M = (P, \Sigma_R, \Sigma_W)$ where*

- **read-access** *labels $\alpha \in \Sigma_R$ do not change the state and for all states there exists at least one outgoing read-label, i.e. for all $p \in S_P$:*
  - *there exists $\alpha \in \Sigma_R$: $p \xrightarrow{\alpha} p$ and*
  - *for all $\alpha \in \Sigma_R$: $p \xrightarrow{\alpha} p' \Rightarrow p' = p$,*
- **write-access** *labels $\alpha \in \Sigma_W$ are non-blocking and cause at least one state change, i.e., for all $\alpha \in \Sigma_W$ there exist $p, p' \in S_P$ with $p \xrightarrow{\alpha} p' \wedge p' \neq p$.*

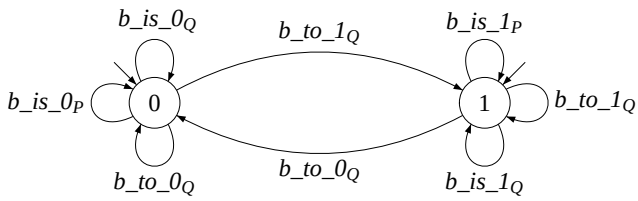


Figure 2.2: A boolean variable b modeled with synchronization labels for read-access by P and read/write-access by Q

We also say that P is a shared variable model over (Σ_R, Σ_W). The set of **read-states** S_R *and* **write-states** S_W *of the variable is the subset of S_P for which there exist incoming transitions with labels in $\Sigma_R \cup \Sigma_W$:*

$$S_R = \{p' \mid \exists \alpha \in \Sigma_R, p : p \xrightarrow{\alpha} p'\}, S_W = \{p' \mid \exists \alpha \in \Sigma_W, p : p \xrightarrow{\alpha} p'\}. \tag{2.4}$$

The shared variable model is called **write-deterministic** *if each write-access label leads to a dedicated state, i.e., for all $\alpha \in \Sigma_W$ there exists a $p'_\alpha \in S_P$ such that for all $p \in S_P$ holds that*

$$p \xrightarrow{\alpha} p'' \Leftrightarrow p'' = p'_\alpha. \tag{2.5}$$

Note that Σ_R and Σ_W are disjunct by definition. If $S_R \neq \emptyset$ then $S_R = S_P$. The following example shall illustrate the concept of a shared variable model with read- and write-access:

Example 2.2. *Figure 2.2 shows the model of a boolean variable for access by two automata P and Q, P with read-access and Q with read/write-access. Synchronization labels $b_is_0_k$ and $b_is_1_k$, $k \in \{P, Q\}$, represent read-access, and $b_to_0_Q$ and $b_to_1_Q$ the write-access by Q. Initially, the value of the variable is undetermined so that both states are initial states. Still, the model is write-deterministic according to Def. 2.14 because after write-access, the state of the variable is determined by the write-access labels.*

It is easy to see from this definition that several modules interfere in reading and writing to variables unless they are nonblocking in a convenient manner. To avoid modelling errors, it is therefore advisable to introduce for each pair of variable of model and accessing module a separate set of labels. The gain from using explicit models of variables is that all automata can freely be abstracted, minimized and analyzed with the methods discussed in the following chapters.

2.5 Related Work

For an overview of discrete event models and their application, refer to [VK88]. An educational tool called LTSA [MK01] is available from Jeff Magee. It accompanies

his book on concurrency [MK99] and provides a graphical user interface for modelling and analyzing both safety (reachability) and liveness (progress) properties of labeled transition systems. A model for shared variables using discrete I/O-automata is given by Lynch in [Lyn96]. There, the set of shared variables is modeled by a single monolithic I/O-automaton to ensure consistency.

Chapter 3

Simulation Relations for Labeled Transition Systems

Simulation relations were introduced by Milner in 1971 for the purpose of comparing programs [Mil71]. They define a preorder $\preceq$ such that $P \preceq Q$ for automata P and Q if every behavior of P finds a match in Q. In such a comparison, P could be, e.g., an implementation and Q a specification, or P a refined and Q a more abstract model. Simulation relations are known to preserve safety properties, e.g., ACTL* formulas [CGP99], and provide a sufficient condition for language inclusion. Classically, simulation relates two systems with identical alphabets. We present a new extension, called Σ-*simulation*, to compare automata with arbitrary alphabets. Our definition contains conventional simulation as a special case when P and Q have the same alphabet, and we can show that it is the largest compositional extension of classic simulation. In 1991, Grumberg and Long presented a framework for compositional reasoning based on a precongruence for Kripke structures [GL91], and a version based on simulation relations can be found in [CGP99].We apply this approach to labeled transition systems and Σ-simulation. For Σ-simulation, any chaos automaton functions as an identity element with respect to parallel composition, which allows us to derive the proposition $P||Q \preceq P$, which is fundamental to compositional reasoning, instead of giving a constructive proof as in [GL91]. The structure of the proofs is shown in Fig. 3.1. Five fundamental properties of simulation, shown as rectangular boxes in Fig. 3.1, are established by constructing witnessing simulation relations. Their algebraic combination lets us establish $P||Q \preceq P$ and three central theorems for compositional reasoning:

- compositionality,
- decomposition of specification, and
- non-circular assume-guarantee reasoning.

This chapter establishes the first two rules. Non-circular assume-guarantee reasoning is left to the next chapter, where we will also deal with circularity.

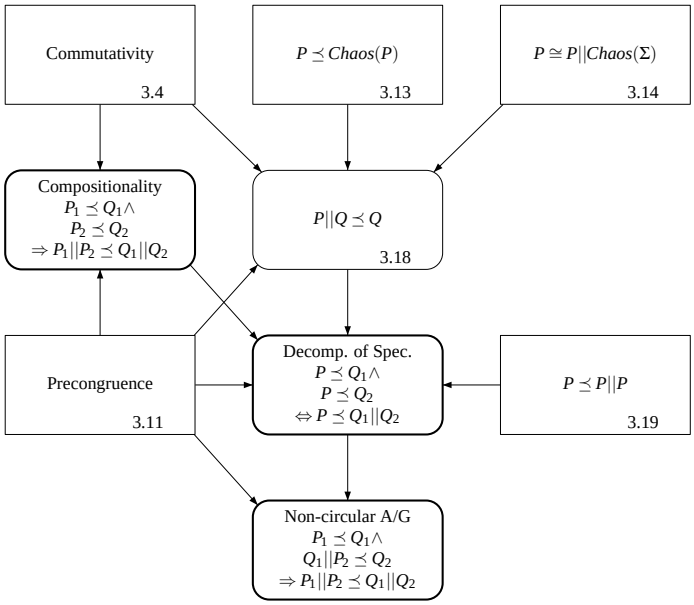


Figure 3.1: Structure of compositional reasoning framework

In the next section, we recall classic simulation, and show that for a preorder compositionality is equivalent to being precongruence, i.e., invariance under composition. Simulation is extended in Sect. 3.2 to accommodate arbitrary alphabets in what we call Σ-simulation. We show that it is compositional, and furthermore the largest compositional extension of simulation to arbitrary alphabets. We derive the building blocks of compositional reasoning and propose a theorem for the decomposition of the specification. In Sect. 3.3 give a brief summary on computational aspects of simulation relations and outline basic algorithms. We conclude the chapter with Sect. 3.4, where we give an overview of related work.

3.1 Simulation

Simulation relations are widely used to show abstraction and refinement between models and specifications. They preserve the branching structure of systems and provide a sufficient condition for language inclusion that can be established with lower complexity. Also, their precongruence properties are ideally suited for compositional reasoning. Intuitively, a simulation relation relates two states in the following way: A state q simulates a state p if, starting from state q, the same, or more, things can happen as when starting from state p. The reader is referred to [CGP99] for a detailed introduction to simulation. In the following, let P and Q be labeled transition sys-

tems $P = (S_P, \Sigma_P, \rightarrow, P_0)$ and $Q = (S_Q, \Sigma_Q, \rightarrow, Q_0)$. We formally define simulation for labeled transition systems following the classic definition by Milner:

Definition 3.1 (Simulation)**.** [Mil71] *For any P and Q with $\Sigma_P = \Sigma_Q$, a relation $R \subseteq S_P \times S_Q$ is a* **simulation relation** *if and only if for all $(p,q) \in R, \alpha \in \Sigma_P, p' \in S_P$ holds:*

$$p \xrightarrow{\alpha} p' \;\Rightarrow\; \exists q' \in S_Q : (q \xrightarrow{\alpha} q' \wedge (p', q') \in R)$$

A state q **simulates** *a state p if there exists a simulation relation R with $(p,q) \in R$, written as $p \preceq q$. Q* **simulates** *P, denoted as $P \preceq Q$, if for every state $p_0 \in P_0$ there exists a state $q_0 \in Q_0$ such that $p_0 \preceq q_0$, i.e., if for any simulation relation R holds $P_0 \subseteq R^{-1}(Q_0)$. The relation R is then called a* **witness** *for $P \preceq Q$.*

A well known, and most fundamental, property of simulation is that it is a *preorder*, i.e., reflexive and transitive [CGP99]. It can therefore be used to define equivalence by mutual simulation. There are two kinds: *Similarity* of LTS P,Q requires that both simulate each other, therefore is the natural equivalence for simulation and preserves the same class of properties. *Bisimulation* on the other hand requires that the same relation witnesses both directions, and so imposes a state-by-state equivalence. It is the most finely grained notion of equivalence commonly used, and preserves all CTL^* properties [CGP99]. Formally, the definitions are as follows:

Definition 3.2 (Similarity, Bisimulation)**.** [CGP99] *Given LTS P, Q, we say that P and Q are* **similar**, *written as $P \simeq Q$, if and only if $P \preceq Q$ and $Q \preceq P$. A relation $R \subseteq S_P \times S_Q$ is a* **bisimulation relation** *if and only if R witnesses $P \preceq Q$ and R^{-1} witnesses $Q \preceq P$. P and Q are* **bisimilar**, *written as $P \cong Q$, if and only if there exists such a relation.*

Note that for similarity, the simulation relation witnessing $P \preceq Q$ can be entirely different from the one witnessing $Q \preceq P$. Simulation is a necessary condition for similarity, and similarity is necessary for bisimulation [1]:

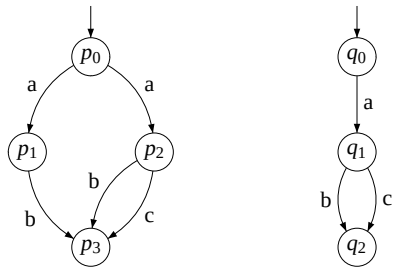
Proposition 3.3. [Par81] *For all P, Q, $P \cong Q \Rightarrow P \simeq Q \Rightarrow P \preceq Q$.*

Note that if there exist simulation relations $R_{P \preceq Q}$ for $P \preceq_{\approx} Q$ and $R_{Q \preceq P}$ for $Q \preceq_{\approx} P$, it is a sufficient condition for bisimulation that $R_{P \preceq Q} = R^{-1}_{Q \preceq P}$, but not a necessary one. The following classic example, which can be found, e.g., in [CGP99, p.173], illustrates the difference between similarity and bisimulation:

Example 3.1. *To illustrate the difference between similarity and bisimulation, consider the labeled transition systems P and Q in Fig. 3.2. Simulation relations for $P \preceq Q$ and $Q \preceq P$ exist, see Table 3.1, and therefore $P \simeq Q$. There exists no bisimulation relation between P and Q because location* 1 *in P has no bisimilar location in Q, so $P \ncong Q$.*

A rather practical property of parallel composition is its commutativity down to structural isomorphism [GL91]:

[1] A unified hierarchy for equivalences of concurrent systems has been established by van Glabbeek [Gla90]. See [HS96] for a survey of behavioral equivalences and [SIRS96] for a survey of their complexity.



(a) P (b) Q

Figure 3.2: Similar, yet not bisimilar, labeled transition systems

Table 3.1: Simulation relations for Ex. 3.1

(a) $R_{P \preceq Q}$

$P \backslash Q$	q_0	q_1	q_2
p_0	•	–	–
p_1	–	•	–
p_2	–	•	–
p_3	•	•	•

(b) $R_{Q \preceq P}$

$Q \backslash P$	p_0	p_1	p_2	p_3
q_0	•	–	–	–
q_1	–	–	•	–
q_2	•	•	•	•

Proposition 3.4. *For any P, Q holds $P||Q \cong Q||P$.*

Proof. Let $R = \{(p,q) \mid (q,p) \in S_{Q||P}\}$. It directly follows from the definition of parallel composition that any transition $(p,q) \xrightarrow{\alpha}_{P||Q} (p',q')$ entails a corresponding transition $(q,p) \xrightarrow{\alpha}_{Q||P} (q',p')$ with $((p',q'),(q',p')) \in R$. Since every initial state (p_0,q_0) of $P||Q$ has a match (q_0,p_0) in the initial states of $Q||P$, and $((p_0,q_0),(q_0,p_0)) \in R$, R is a witness for $P||Q \cong Q||P$. □

With Prop. 3.3 it is straightforward that the commutativity extends to bisimulation, similarity and simulation:

Corollary 3.5. *With Props. 3.4 and 3.3 holds $P||Q \simeq Q||P$ and $P||Q \preceq Q||P$.*

Most systems of practical interest can be divided into a set of subsystems, e.g., by physical aspects or function. A compositional approach to modeling and analysis of such a system is based only on the descriptions of subsystems, without further information about the composed system, a concept that was first formalized by Gottlob Frege in 1923 [Fre23]. In the context of automata and simulation, compositionality is captured by the following definition (for related work see Sect. 3.4):

Definition 3.6 (Compositionality)**.** [SS01, AGLS01] *A relation $\sim$ between automata is* **compositional** *if $P_1 \sim Q_1$ and $P_2 \sim Q_2$ implies $P_1||P_2 \sim Q_1||Q_2$.*

For practical purposes it is reasonable to require the composition operator to be commutative and associative. The following proposition relates compositionality to invariance under composition[2]. It motivates the requirements for a compositional simulation relation in the next section:

Proposition 3.7. *For a commutative composition operator $||$, a preorder $\sim$ is compositional if and only if it is* **invariant under composition***, i.e., if*

$$P_1 \sim Q_1 \;\Rightarrow\; P_1||P_2 \sim Q_1||P_2. \tag{3.1}$$

Proof. For the sufficient direction, assume that $\sim$ is compositional. Since $\sim$ is reflexive, $P_2 \sim P_2$ holds. Using the definition of compositionality, $P_1 \sim Q_1$ and $P_2 \sim P_2$ imply $P_1||P_2 \sim Q_1||P_2$, and so $\sim$ is invariant under composition. For the necessary direction, assume that $\sim$ is invariant under composition. Then $P_1 \sim Q_1$ implies $P_1||P_2 \sim Q_1||P_2$. Also, $P_2 \sim Q_2$ and commutativity of composition implies $Q_1||P_2 \sim Q_1||Q_2$, and by transitivity $P_1||P_2 \sim Q_1||Q_2$. □

A preorder that is invariant under a composition operator is called a ***precongruence*** with respect to that operator. It is a well known fact that simulation is a precongruence with respect to parallel composition, therefore readily applied in compositional reasoning. We recall the formal statement:

Proposition 3.8. [CGP99] *Simulation is a precongruence with respect to parallel composition over the set of labeled transition systems with identical alphabets.*

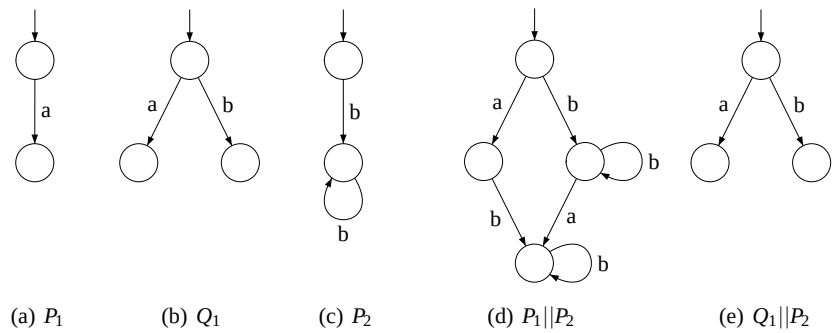
We omit the proof because this is a well established result. Also, we will extend simulation in Sect. 3.2, and the proof of the corresponding proposition on precongruence in that section, Prop. 3.11, contains Prop. 3.8 as a special case.

3.2 Σ-Simulation

In the following, we extend the classic definition of simulation to compare systems with arbitrary alphabets. Because we intend to use this extension for compositional reasoning, Prop. 3.7 requires that it to be a precongruence, i.e., a preorder that is invariant under composition. If we simply apply classic simulation to automata with differing alphabets, it is not preserved under parallel composition, as illustrated by the following example:

Example 3.2. *Consider the LTSs P_1, Q_1 and P_2 shown in Fig. 3.3, with alphabets $\Sigma_{P_1} = \{a\}$, $\Sigma_{Q_1} = \{a,b\}$ and $\Sigma_{P_2} = \{b\}$. Q_1 is blocking on a label that is not in the alphabet of P_1, and so $P_1||P_2 \not\preceq Q_1||P_2$ even though $P_1 \preceq Q_1$ holds.*

[2]Invariance under parallel composition is part of the compositional reasoning framework presented in [GL91]. It is also required by Alur and Henzinger in [AH97] in their modular framework for hybrid systems.



(a) P_1 (b) Q_1 (c) P_2 (d) $P_1||P_2$ (e) $Q_1||P_2$

Figure 3.3: Justification of non-blocking requirement

As a consequence of this deficiency of classic simulation, we impose additional restrictions. Consider simulation for LTS P,Q, i.e., $P \preceq Q$. For labels α that are in the alphabets of both P and Q we retain Def. 3.1 of simulation, and call it condition (i). The remaining cases, in which the label is not in both alphabets, are:

(ii) $\alpha \in \Sigma_Q \backslash \Sigma_P$: As illustrated by Ex. 3.2, Q should be non-blocking on all labels that are not in Σ_P. This is necessary, amongst other reasons, to preserve non-blocking, since if a label α is not in Σ_P, P is non-blocking on α by definition[3].

(iii) $\alpha \in \Sigma_P \backslash \Sigma_Q$: Labels that are not in Σ_Q are allowed as long as the transitions don't lead to violating states. Therefore the target states of such transitions must be inside the relation.

As far as we know, this extension is novel, and we call it Σ-simulation. The conditions are motivated formally at the end of Sect. 3.2.2 by showing that any relation that shall imply simulation in a manner that is invariant under composition must be a subset of a Σ-simulation.[4] We define Σ-simulation for an arbitrary alphabet Σ', and write $\preceq_\Sigma$ if Σ' is the union of the alphabets on both sides of the inequality.[5]

Definition 3.9 (Σ-Simulation, Σ–Similarity, Σ-Bisimulation)**.** *Given an alphabet Σ' and LTSs P, Q, a relation $R \subseteq S_P \times S_Q$ is a Σ'-**simulation relation** if and only if for all $(p,q) \in R, \alpha \in \Sigma', p' \in S_P$ holds:*

(i) $\alpha \in \Sigma_P \cap \Sigma_Q$ *and* $p \xrightarrow{\alpha} p' \;\Rightarrow\; \exists q' \in S_Q : (q \xrightarrow{\alpha} q' \wedge (p',q') \in R)$

[3]Strictly speaking, non-blocking is not required of all states in Q, but only of those in the witnessing simulation relation. This is relevant if Q is non-deterministic or has unreachable states.

[4]Jonsson defined in [Jon94] simulation for $\Sigma_Q \subseteq \Sigma_P$, but without including any additional conditions. As we show in Sect. 3.2.2, this is not invariant under composition and, consequently, not compositional.

[5]The case where $\Sigma' \subset \Sigma_P \cup \Sigma_Q$ is useful for dealing with uncontrollable and unobservable transitions, which are not used in this thesis.

(ii) $\alpha \in \Sigma_Q \backslash \Sigma_P$ *and* $\exists q' \in S_Q : (q \xrightarrow{\alpha} q' \wedge (p,q') \in R)$

(iii) $\alpha \in \Sigma_P \backslash \Sigma_Q$ *and* $p \xrightarrow{\alpha} p' \;\Rightarrow\; (p',q) \in R$.

*A state q Σ'-**simulates** a state p if there exists a Σ'-simulation relation R with $(p,q) \in R$, written as $p \preceq_{\Sigma'} q$. Q Σ'-**simulates** P, denoted as $P \preceq_{\Sigma'} Q$, if for every state $p_0 \in P_0$ there exists a state $q_0 \in Q_0$ such that $p_0 \preceq_{\Sigma'} q_0$, i.e., if for any Σ'-simulation relation R holds $P_0 \subseteq R^{-1}(Q_0)$. The relation R is then called a* **witness** *for $P \preceq_{\Sigma'} Q$. Let $\Sigma = \Sigma_P \cup \Sigma_Q$. Σ-similarity and Σ-bisimulation are extended from Def. 3.1 in the straightforward manner, and denoted with $\simeq_{\Sigma'}$, respectively $\cong_{\Sigma'}$.*

It follows directly from the definitions that simulation and Σ-simulation are identical if both sides of the inequality $P \preceq Q$ have the same alphabet, so that simulation is a special case of Σ-simulation:

Proposition 3.10. *For any P,Q with $\Sigma_P = \Sigma_Q$ holds $P \preceq Q \;\Leftrightarrow P \preceq_\Sigma Q$.*

If it is clear from the context which notion of simulation is required, we will omit the distinction between $\preceq$ and $\preceq_\Sigma$, and simply write $\preceq$. With its three conditions, Σ-simulation is more cumbersome than simulation for low level proofs, i.e., proofs that involve the construction of a witnessing simulation relation. High-level proofs, however, can be simpler and more flexible because of the ability to compare arbitrary LTS, as will be evident in the following sections. Particularly interesting is the ability to compose and decompose specifications, which will be shown in Sect. 3.2.2. Also, specification models can be slightly smaller, since non-blocking labels do not have to be modeled by self-loops. In Sect. 3.2.1 we will introduce *Chaos*-LTS to close the gap between simulation and Σ-simulation and show that they can be expressed in either form. It remains to be shown that the goal of this section, to define simulation with precongruence properties for arbitrary alphabets, was attained:

Proposition 3.11. *With respect to parallel composition, Σ-Simulation is a precongruence on the class of labeled transition systems.*

Proof. The proof is carried out by constructing witnessing simulation relations. It is similar to the corresponding proof for simulation given in [GL91], of which a didactically refined version can be found in [CGP99, p. 176,188].

Reflexivity:
$R = \{(p,p)\}$ is a witness that $\preceq$ is reflexive, since only condition (i) applies and $p \xrightarrow{\alpha} p' \Rightarrow (p \xrightarrow{\alpha} p' \wedge (p',p') \in R)$. It is easy to see that $P_0 \subseteq R^{-1}(P_0)$, so R is a witness for $P \preceq_\Sigma P$.

Transitivity:
Let R_1 and R_2 be witnesses for $P \preceq_\Sigma Q$ and $Q \preceq_\Sigma S$ respectively. We will show that $R = \{(p,s) | \exists q : (p,q) \in R_1 \wedge (q,s) \in R_2\}$ is a witnessing simulation relation. Consider p,q,s with $(p,q) \in R_1$ and $(q,s) \in R_2$.

(i) Let $\alpha \in \Sigma_P \cap \Sigma_S$. Then R is not a simulation relation if $p \xrightarrow{\alpha} p'$ and there exists no $s' : s \xrightarrow{\alpha} s'$ with $(p', s') \in R$. It will be shown that this is never the case.

- If $\alpha \in \Sigma_Q$, then case (i) applies to R_1 and there exists a q' with $(q \xrightarrow{\alpha} q' \wedge (p', q') \in R_1)$. Since $\alpha \in \Sigma_S$, case (i) applies also for R_2 and s' exists with $(q', s') \in R_2$. With $(p', q') \in R_1$ follows that $(p', s') \in R$.
- If $\alpha \notin \Sigma_Q$, then case (iii) applies to R_1 and therefore $(p', q) \in R_1$. For R_2, case (ii) applies and therefore s' exists with $(q, s') \in R_2$. Again, $(p', s') \in R$.

(ii) Let $\alpha \in \Sigma_S \backslash \Sigma_P$. It must be shown that s' exists with $s \xrightarrow{\alpha} s'$ and $(p, s') \in R$.

- If $\alpha \in \Sigma_Q$, then case (ii) applies to R_1 and there exists a q' with $(q \xrightarrow{\alpha} q' \wedge (p, q') \in R_1)$. Since $\alpha \in \Sigma_S$, case (i) applies to R_2 and s' exists with $s \xrightarrow{\alpha} s'$ and $(q', s') \in R_2$. With $(p, q') \in R_1$ follows that $(p, s') \in R$.
- If $\alpha \notin \Sigma_Q$, then case (ii) applies to R_2, and s' exists with $s \xrightarrow{\alpha} s'$ and $(q, s') \in R_2$. Since $(p, q) \in R_1$ it follows that $(p, s') \in R$.

(iii) Let $\alpha \in \Sigma_P \backslash \Sigma_S$. It must be shown that $p \xrightarrow{\alpha} p' \Rightarrow (p', s) \in R$.

- If $\alpha \in \Sigma_Q$, then case (i) applies for R_1 and there exists a q' with $(q \xrightarrow{\alpha} q' \wedge (p', q') \in R_1)$. Since $\alpha \notin \Sigma_S$, case (iii) applies also for R_2 and $(q', s) \in R_2$. With $(p', q') \in R_1$ follows that $(p', s) \in R$.
- If $\alpha \notin \Sigma_Q$, then the only option is (iii) and therefore $(p', q) \in R_1$. In combination with $(q, s) \in R_2$ this implies $(p', s) \in R$, as required.

By definition of simulation it holds that $P_0 \subseteq R_1^{-1}(Q_0)$ and $Q_0 \subseteq R_2^{-1}(S_0)$. So for every $p_0 \in P_0$ there is a $q_0 \in Q_0$ with $(p_0, q_0) \in R_1$, also a $s_0 \in S_0$ with $(q_0, s_0) \in R_2$. With $(p_0, s_0) \in R$ it follows that $P_0 \subseteq R^{-1}(S_0)$, and consequently R witnesses $P \preceq_\Sigma S$.

Invariance under Composition:
It must be shown that if R_0 is a simulation relation for $P \preceq Q$, then there exists a simulation relation for $P||S \preceq_\Sigma Q||S$. We will show that $R = \{((p,s),(q,s)) | (p,q) \in R_0\}$ is a witnessing simulation relation. Assume that $(p,q) \in R_0$.

(i) $\alpha \in (\Sigma_P \cup \Sigma_S) \cap (\Sigma_Q \cup \Sigma_S)$: It must be shown that if $(p,s) \xrightarrow{\alpha} (p',s')$ there exists (q',s') with $(q,s) \xrightarrow{\alpha} (q',s')$ and $((p',s'),(q',s')) \in R$.

(a) $\alpha \in \Sigma_P$: Then $(p,s) \xrightarrow{\alpha} (p',s')$ implies $p \xrightarrow{\alpha} p'$ and therefore $(p,q) \in R_0$ implies that a q' exists with $q \xrightarrow{\alpha} q'$ and $(p',q') \in R_0$, and by parallel composition there is a transition $(q,s) \xrightarrow{\alpha} (q',s')$, and $((p',s'),(q',s')) \in R$.

(b) $\alpha \notin \Sigma_P \cup \Sigma_Q$: Then $(p,s) \xrightarrow{\alpha} (p',s')$ implies $s \xrightarrow{\alpha} s'$ and $p = p'$. Then there is also a transition $(q,s) \xrightarrow{\alpha} (q',s')$ with $q = q'$. Since $(p,q) \in R_0$ it follows that $(p',q') \in R_0$ and $((p',s'),(q',s')) \in R$.

(c) $\alpha \in \Sigma_Q \backslash \Sigma_P$: By parallel composition it holds that $p = p'$. Then case (ii) applies for R_0 and q' exists with $q \xrightarrow{\alpha} q'$ and $(p,q') \in R_0$ and therefore there is a transition $(q,s) \xrightarrow{\alpha} (q',s')$ and it holds that $((p,s'),(q',s')) \in R$.

(ii) $\alpha \in (\Sigma_Q \cup \Sigma_S) \backslash (\Sigma_P \cup \Sigma_S) = \Sigma_Q \backslash (\Sigma_P \cup \Sigma_S)$: It must be shown that there exists (q',s') with $(q,s) \xrightarrow{\alpha} (q',s')$ and $((p,s),(q',s')) \in R$. Since $\alpha \notin \Sigma_P$, $(p,q) \in R_0$ implies with case (ii) that q' exists with $(p,q') \in R_0$. Since $\alpha \notin \Sigma_S$, $s' = s$ and therefore $((p,s),(q',s')) \in R$.

(iii) $\alpha \in (\Sigma_P \cup \Sigma_S) \backslash (\Sigma_Q \cup \Sigma_S) = \Sigma_P \backslash (\Sigma_Q \cup \Sigma_S)$: It must be shown that if $(p,s) \xrightarrow{\alpha} (p',s')$ then it holds that $((p',s'),(q,s)) \in R$. By parallel composition, $(p,s) \xrightarrow{\alpha} (p',s')$ implies a transition $p \xrightarrow{\alpha} p'$ and $s = s'$. Therefore $(p,q) \in R_0$ implies with case (iii) for R_0 that $(p',q) \in R_0$. For any $s = s'$ holds $((p',s'),(q,s)) \in R$.

From the premise it follows that for every $p_0 \in P_0$ there exists some $q_0 \in Q_0$ with $(p_0,q_0) \in R_0$. With the definition of parallel composition it follows that for every (p_0,s_0) in the initial states of $P||S$ and $q_0 \in Q_0$, (q_0,s_0) is in the initial states of $Q||S$. From the definition of R and R_0 it follows that for every $p_0 \in P_0$ there exists such a $q_0 \in Q_0$, and consequently $P_0 \times S_0 \subseteq R^{-1}(Q_0 \times S_0)$ holds, which makes R a witness for $P||S \preceq_\Sigma Q||S$. □

3.2.1 Simulation, Σ-Simulation and Chaos

Proposition 3.10 states that simulation is a special case of Σ-simulation. In the following we will express Σ-simulation in terms of simulation, which allows us to effortlessly switch between the two types of simulation. Since the alphabets on both sides of an inequality $P \preceq Q$ need to be equal, we add additional labels by composing both sides with a so-called chaos system that creates the appropriate transitions [6]. The lemmata established in this section allow the manipulation of inequalities involving chaos automata and are essential building blocks of our framework for compositional reasoning. In every state of a chaos system, there is a transition for every label in the alphabet:

Definition 3.12 (Chaos)**.** *A* **chaos** *system over the alphabet* Σ_0 *is the labeled transition system* $Chaos(\Sigma_0) = (\{chaos\}, \Sigma_0, \rightarrow, \{chaos\})$ *with* $\rightarrow \; = \; \{chaos\} \times \Sigma_0 \times \{chaos\}$. *For a labeled transition system* $P = (S_P, \Sigma_P, \rightarrow, P_0)$*, let* $Chaos(P) := Chaos(\Sigma_P)$.

A chaos system Σ-simulates every other LTS, making it an upper bound on any set of LTSs:

[6] The inspiration for the chaos system comes from the theory of Communicating Sequential Processes by C.A.R. Hoare, see [Hoa85], pp.126. There, a CHAOS process over an alphabet A can nondeterministically produce any action from A at any time. In [Lon93], an automaton $\top(A)$, similar in spirit to the *Chaos* automaton here, is defined for Kripke structures [Lon93, p. 71] and used to relate structures with different alphabets.

Lemma 3.13. *For any P and any alphabet Σ', $P \preceq_\Sigma Chaos(\Sigma')$.*

Proof. Let $R = \{(p, chaos) | p \in S_P\}$. We show that the conditions (i)-(iii) of Def. 3.9 are fulfilled, so that R is a Σ-simulation relation:

(i) $\alpha \in \Sigma_P \cap \Sigma'$: By Def. 3.12, a transition $chaos \xrightarrow{\alpha} chaos$ exists for any $\alpha \in \Sigma_P \cap \Sigma'$. Therefore, for any p' with $p \xrightarrow{\alpha} p'$ there exists a $q' = chaos$ with $(p', q') \in R$.

(ii) $\alpha \in \Sigma' \setminus \Sigma_P$: By Def. 3.12, a transition $chaos \xrightarrow{\alpha} chaos$ exists for any $\alpha \in \Sigma' \setminus \Sigma_P$. Therefore, there always exists a $q' = chaos$ with $(p, q') \in R$.

(iii) $\alpha \in \Sigma_P \setminus \Sigma'$: For any $(p, q) \in R$ and any p' with $p \xrightarrow{\alpha} p'$ it holds that $(p', q) \in R$.

For all $p_0 \in P_0$ holds that $(p_0, chaos) \in R$, so $P_0 \subseteq R^{-1}(chaos)$ holds, and R is a witness for $P \preceq_\Sigma Chaos(\Sigma')$. □

It follows from Lemma 3.13 and its proof that the set of all *Chaos*-automata form an equivalence class with respect to bisimulation, and therefore also with respect to similarity and simulation. The composition of an arbitrary automaton P with a chaos automaton is in the same equivalence class with respect to bisimulation as P. This is expressed by the following lemma:

Lemma 3.14. *For any P and any alphabet Σ', $P \cong_\Sigma P || Chaos(\Sigma')$.* [7]

Proof. We will show that $R = \{(p, (p, chaos))\}$ is a witness for $P \preceq_\Sigma P || Chaos(\Sigma')$, and that R^{-1} witnesses $P || Chaos(\Sigma') \preceq_\Sigma P$. To prove the first claim, we demonstrate that the conditions (i)-(iii) of Def. 3.9 are fulfilled by R, and that the initial states are properly contained in R:

(i) $\alpha \in \Sigma_P \cap (\Sigma_P \cup \Sigma') = \Sigma_P$:

- $\alpha \in \Sigma_P \cap \Sigma'$: By Def. 3.12, a transition $chaos \xrightarrow{\alpha} chaos$ exists for any $\alpha \in \Sigma_P \cap \Sigma'$. Therefore, for any p' with $p \xrightarrow{\alpha} p'$ there exists a $q' = (p', chaos)$ with $(p', q') \in R$ and $(p, chaos) \xrightarrow{\alpha} (p', chaos)$.
- $\alpha \in \Sigma_P \setminus \Sigma'$: For any $(p, q) \in R$ and any p' with $p \xrightarrow{\alpha} p'$ it holds that $q = (p, chaos)$ and $\alpha \in \Sigma_P \setminus \Sigma'$ implies by the definition of parallel composition an independent transition $(p, chaos) \xrightarrow{\alpha} (p', chaos)$. Therefore, there always exists a $q' = (p', chaos)$ with $(p', q') \in R$.

(ii) $\alpha \in (\Sigma_P \cup \Sigma') \setminus \Sigma_P = \Sigma' \setminus \Sigma_P$: By Def. 3.12, a transition $chaos \xrightarrow{\alpha} chaos$ exists for any $\alpha \in \Sigma' \setminus \Sigma_P$. By the definition of parallel composition, $\alpha \in \Sigma' \setminus \Sigma_P$ implies that there is a transition $(p, chaos) \xrightarrow{\alpha} (p, chaos)$ in $P || Chaos(\Sigma')$. Therefore, there always exists a $q' = (p, chaos)$ with $(p, q') \in R$.

[7] See also [Lon93, p. 77] for a similar application to Kripke structures.

(iii) $\alpha \in \Sigma_P \setminus (\Sigma_P \cup \Sigma') = \emptyset$ does not occur.

Since for all $p_0 \in P_0$ holds that $(p_0, (p_0, chaos)) \in R$, and $(p_0, chaos)$ is an initial state of $P||Chaos(\Sigma')$, it follows that R is a simulation relation that witnesses $P \preceq_\Sigma P||Chaos(\Sigma')$. Now we show in a similar manner that $P||Chaos(\Sigma') \preceq_\Sigma P$ holds:

(i) $\alpha \in (\Sigma_P \cup \Sigma') \cap \Sigma_P = \Sigma_P$:

- $\alpha \in \Sigma_P \cap \Sigma'$: By Def. 3.12, a transition $chaos \xrightarrow{\alpha} chaos$ exists for any $\alpha \in \Sigma_P \cap \Sigma'$. By definition of parallel composition, a transition in $P||Chaos(\Sigma')$ occurs if and only if there is a transition $p \xrightarrow{\alpha} p'$. It immediately follows that there exists a p' with $((p', chaos), q') \in R^{-1}$.
- $\alpha \in \Sigma_P \setminus \Sigma'$: By the definition of parallel composition there is a transition $(p, chaos) \xrightarrow{\alpha} (p', chaos)$ if and only if $p \xrightarrow{\alpha} p'$. Therefore, there always exists a p' with $((p', chaos), p') \in R^{-1}$.

(ii) $\alpha \in \Sigma_P \setminus (\Sigma_P \cup \Sigma') = \emptyset$ does not occur.

(iii) $\alpha \in (\Sigma_P \cup \Sigma') \setminus \Sigma_P = \Sigma' \setminus \Sigma_P$: By Def. 3.12, a transition $chaos \xrightarrow{\alpha} chaos$ exists for any $\alpha \in \Sigma' \setminus \Sigma_P$. By the definition of parallel composition, $\alpha \in \Sigma' \setminus \Sigma_P$ implies that there is a transition $(p, chaos) \xrightarrow{\alpha} (p, chaos)$ in $P||Chaos(\Sigma')$, and $((p, chaos), p) \in R^{-1}$.

For all $(p_0, chaos)$ with $p_0 \in P_0$, i.e., all initial states of $P||Chaos(\Sigma')$, holds that $((p_0, chaos), p_0) \in R^{-1}$ and p_0 is an initial state of $P||Chaos(\Sigma')$. Therefore, R^{-1} is a simulation relation that witnesses $P||Chaos(\Sigma') \preceq_\Sigma P$ and R is a Σ-bisimulation relation that witnesses $P \cong_\Sigma P||Chaos(\Sigma')$. □

From Lemma 3.14 and the transitivity of Σ-simulation, it follows that the chaos automaton is an identity element for Σ-simulation. For any alphabet, it can be added and omitted at will:

Theorem 3.15. *For any P, Q and arbitrary alphabets Σ_1, Σ_2 holds:*

$$P \preceq_\Sigma Q \;\Leftrightarrow\; P||Chaos(\Sigma_1) \preceq_\Sigma Q||Chaos(\Sigma_2). \tag{3.2}$$

Proof. Assume that $P \preceq_\Sigma Q$ holds. With Lemma 3.14 it follows that $P||Chaos(\Sigma_1) \preceq_\Sigma P$, and by transitivity follows $P||Chaos(\Sigma_1) \preceq_\Sigma Q$. Similarly, Lemma 3.14 implies that $Q \preceq_\Sigma Q||Chaos(\Sigma_2)$, and by transitivity we get $P||Chaos(\Sigma_1) \preceq_\Sigma Q||Chaos(\Sigma_2)$. Assume that $P||Chaos(\Sigma_1) \preceq_\Sigma Q||Chaos(\Sigma_2)$ holds. With Lemma 3.14 it follows that $P \preceq_\Sigma P||Chaos(\Sigma_1)$, and by transitivity follows $P \preceq_\Sigma Q||Chaos(\Sigma_2)$. Similarly, Lemma 3.14 implies that $Q||Chaos(\Sigma_2) \preceq_\Sigma Q$, and by transitivity we get $P \preceq_\Sigma Q$. □

Theorem 3.15 immediately allows us to formulate Σ-simulation in terms of classical simulation by composing chaos automata with both sides, which balances the alphabets:

Corollary 3.16. *For any P,Q, $P \preceq_\Sigma Q$ if and only if $P||Chaos(Q) \preceq Q||Chaos(P)$.* [8]

Proof. With Theorem 3.15, $P \preceq_\Sigma Q$ is equivalent to $P||Chaos(Q) \preceq_\Sigma Q||Chaos(P)$. The alphabet on both sides is equal to $\Sigma_P \cup \Sigma_Q$, so that with Prop. 3.10 this is equivalent to $P||Chaos(Q) \preceq Q||Chaos(P)$. □

According to Corollary 3.16, any of the following proofs or algorithms for Σ-simulation can be applied to classic simulation by balancing the alphabets with chaos automata. We can now easily show that Σ-simulation is an upper bound on any relation that implies simulation invariant under composition:

Theorem 3.17 (Maximality of Σ-simulation)**.** *For any LTS P,Q and a relation R, it holds that $(p,q) \in R$ implies $(p,s) \preceq (q,s)$ for any state s of an arbitrary LTS S with $\Sigma_P \cup \Sigma_Q \subseteq \Sigma_S$ if and only if $(p,q) \in R \Rightarrow p \preceq_\Sigma q$.*

Proof. For the sufficient condition, assume that $(p,q) \in R \Rightarrow p \preceq_\Sigma q$. With Prop. 3.11 it follows that $(p,s) \preceq_\Sigma (q,s)$. Since the alphabets on both sides are equal, Prop. 3.10 yields that $(p,s) \preceq (q,s)$. For the necessary condition, assume $(p,q) \in R \Rightarrow (p,s) \preceq (q,s)$. Let's consider $(p,s) \preceq (q,s)$. Because the alphabets on both sides are equal, Prop. 3.10 yields that $(p,s) \preceq_\Sigma (q,s)$. Since the assumption holds for any S, let $S = Chaos(\Sigma_P \cup \Sigma_Q)$. With Theorem 3.15, $(p,chaos) \preceq_\Sigma (q,chaos) \Leftrightarrow p \preceq_\Sigma q$. So for this choice of S, $(p,q) \in R \Rightarrow (p,s) \preceq (q,s)$ implies $p \preceq_\Sigma q$. If R must imply $(p,s) \preceq (q,s)$ for any S, it cannot be larger than $\preceq_\Sigma$. □

We can conclude that Σ-simulation is the largest, or most permissive, preorder that implies simulation and is invariant under composition.

3.2.2 Decomposition of the Specification

We proceed with the remaining fundamental propositions in the compositional framework of Fig. 3.1. They culminate in a theorem showing that when specifications, i.e., the right side of an inequality $P \preceq Q$, are composed and decomposed, the composition operator on the right hand side of the simulation inequality behaves like a logical AND operation. The sufficient direction of the decomposition is based on the following fundamental property of simulation:

Proposition 3.18. [GL91] *For any P,Q, $P||Q \preceq_\Sigma Q$.*

Proof. With Lemma 3.13 holds $P \preceq_\Sigma Chaos(\Sigma_P)$, witnessed by $R' = \{(p,chaos)\}$, and with invariance under composition follows $P||Q \preceq_\Sigma Chaos(\Sigma_P)||Q$, witnessed by $R'' = \{((p,q),(chaos,q))|(p,chaos) \in R'\} = \{((p,q),(chaos,q))\}$. Lemma 3.14 and commutativity yields $Chaos(\Sigma_P)||Q \preceq_\Sigma Q$, witnessed by $R = \{((chaos,q),q)\}$, and the conclusion follows by transitivity with $R = \{((p,q),q)\}$ as a witness. □

[8] A similar approach is used in [Lon93, p. 120] to compare Kripke structures of different alphabets.

The necessary direction of the decomposition relies on an equally fundamental proposition, which shows that a labeled transition system can be arbitrarily composed with itself without effect on the preorder:

Proposition 3.19 (Idempotency)**.** [GL91] *For any P holds that $P \preceq_\Sigma P||P$.*

Proof. Let $R = \{(p,(p,p)) \mid p \in S_P\}$. For any $p' \in S_P, \alpha \in \Sigma_P$ with $p \xrightarrow{\alpha}_P p'$ there trivially exists a transition $(p,p) \xrightarrow{\alpha}_{P||P} (p',p')$, and $(p',(p',p')) \in R$. Consequently, R is a simulation relation. Since for every $p_0 \in S_{P0}$ there exists a state (p_0,p_0) in the initial states of $P||P$, R is a witness for $P \preceq_\Sigma P||P$. □

Using these propositions and the precongruence properties of simulation, we can show that the specification can be decomposed arbitrarily.

Theorem 3.20 (Decomposition of Specification)**.** *For any P, Q_1, Q_2, $P \preceq_\Sigma Q_1||Q_2$ if and only if $P \preceq_\Sigma Q_1$ and $P \preceq_\Sigma Q_2$.*

Proof. Assume that $P \preceq_\Sigma Q_1||Q_2$. According to Prop. 3.18 it holds that $Q_1||Q_2 \preceq_\Sigma Q_1$ and $Q_1||Q_2 \preceq_\Sigma Q_2$. From transitivity follows the sufficient direction of the theorem. For the necessary direction, assume that $P \preceq_\Sigma Q_1$ and $P \preceq_\Sigma Q_2$. With compositionality it holds that $P||P \preceq_\Sigma Q_1||Q_2$, and with Prop. 3.19 and transitivity follows $P \preceq_\Sigma Q_1||Q_2$. □

The generalization to an arbitrary number of LTSs follows immediately by recursion. Together with Corollary 3.16 Theorem 3.20 also provides a decomposition for the simulation of Def. 3.1:

Corollary 3.21. *For any P, Q_1 and Q_2 with $\Sigma_P = \Sigma_{Q_1} \cup \Sigma_{Q_2}$ it holds that $P \preceq Q_1||Q_2$ if and only if $P \preceq Q_1||Chaos(P)$ and $P \preceq Q_2||Chaos(P)$.*

This concludes the decomposition of the specification, which allows to verify smaller sub-specifications instead of a single large one. The decomposition of the system, and using proofs on parts of the system instead of a single one, will be the subject of Chapter 4 on assume/guarantee-reasoning.

3.3 Computing Simulation Relations

The computability of simulation relations as the least fixed-point of a monotonic operator follows from Tarski's fixed-point theorem [Tar55]. Here we recall a direct standard proof :

Definition 3.22 (Simulation up to n)**.** [CGP99] *For each $n \geq 0$, the relation $\preceq_n \subseteq S_P \times S_Q$ is inductively defined as follows:*

1. $p \preceq_0 q \; \forall (p,q) \in S_P \times S_Q$,

procedure *CheckSimulation*
Input: labeled transition systems P,Q
Output: a simulation relation R

$R := S_P \times S_Q$
$R' := \emptyset$
while $R \neq R'$ **do**
$R' := R$
$R := R \backslash \{(p,q) | p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p',q') \in R\}$
od

Figure 3.4: Algorithm for checking simulation of labeled transition systems

2. $p \preceq_{n+1} q \Leftrightarrow \forall \alpha \in \Sigma_P \cap \Sigma_Q : p \xrightarrow{\alpha} p' \;\Rightarrow\; \exists q' : q \xrightarrow{\alpha} q' \wedge p' \preceq_n q'$.

Proposition 3.23. [CGP99] *For all* $n \geq 0$,

(i) $\preceq \subseteq \preceq_n$.

(ii) $\preceq_{n+1} \subseteq \preceq_n$.

(iii) *If* $\preceq_n = \preceq_{n+1}$, *then* $\preceq_n = \preceq$.

It follows immediately from Def. 3.22 and Prop. 3.23 that $\preceq_{n+1}$ can be constructed by trimming $\preceq_n$:

$$\preceq_{n+1} = \preceq_n \cap \left\{ (p,q) \in \preceq_n | p \xrightarrow{\alpha} p' \Rightarrow q' : q \xrightarrow{\alpha} q' \wedge (p',q') \in \preceq_n \right\}.$$

This is equivalent to

$$\preceq_{n+1} = \preceq_n \setminus \left\{ (p,q) | p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p',q') \in \preceq_n \right\}$$

This leads to the algorithm in Fig. 3.4. Its correctness is assured by Prop. 3.23 (iii). Termination is guaranteed by Prop. 3.23 (ii) and the finiteness of $S_P \times S_Q$. Simulation can be checked in polynomial time, and algorithms that are quadratic in the size of P and Q have been presented in [HHK95, BP95]. Figure 3.5 shows the same algorithm for Σ-Simulation. As has been pointed out in [WL97], the simulation relation R in the algorithm from Fig. 3.4 can be initialized with $R := reach_{P||Q}$ instead of $R := S_P \times S_Q$.

Proposition 3.24. *If a relation* $R \subseteq S_P \times S_Q$ *witnesses* $P \preceq Q$, *then so does* $R' = R \cap reach_{P||Q}$.

A corresponding algorithm that iterates on a waiting list W is shown in Fig. 3.6. Since R is initialized with the reachable states, W is initialized with R and the set of bad states F could contain the entire W in the first iteration, the space requirement could be $2 \times |reach_{P||Q}|$.

procedure *CheckSigmaSimulation*
Input: labeled transition systems P, Q
Output: a Σ-simulation relation R

$R := S_P \times S_Q$
$R' := \emptyset$
while $R \neq R'$ **do**
$R' := R$
$F_i := \{(p,q) | \exists \alpha \in \Sigma_P \cap \Sigma_Q : \exists p' : p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p',q') \in R\}$
$F_{ii} := \{(p,q) | \exists \alpha \in \Sigma_Q \backslash \Sigma_P : \nexists q' : q \xrightarrow{\alpha} q' \wedge (p,q') \in R\}$
$F_{iii} := \{(p,q) | \exists \alpha \in \Sigma_P \backslash \Sigma_Q : \exists p' : p \xrightarrow{\alpha} p' \wedge (p',q) \notin R\}$
$R := R \backslash (F_i \cup F_{ii} \cup F_{iii})$
od

Figure 3.5: Algorithm for checking Σ-simulation of labeled transition systems

procedure *CheckSimulation2*
Input: labeled transition systems P,Q
Output: a simulation relation R

$R := reach_{P||Q}$
$W := R$
while $W \neq \emptyset$ **do**
$F := \{(p,q) \in W | p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p',q') \in R\}$
$R := R \backslash F$
$W := pre(F) \cap R$
od

Figure 3.6: Algorithm for checking simulation of labeled transition systems with a waiting list and starting from $P||Q$

Figure 3.7 shows an on-the-fly algorithm that simultaneously builds the reachable states and checks for simulation. It constructs the set of reachable states A and a set of bad states B. In the first interior loop, the reachable state space is explored with a waiting list W as long as no bad states are encountered. A bad state is a state that either has no matching successor in Q or all those successors are bad states. In the second interior loop, the bad states are backpropagated. The constriction to reachable states in the waiting list V can significantly reduce the search space, since otherwise both the forward and backwards reachable states of (p_0, q_0) might be visited.

It is not possible to restrict A to states that are not forbidden, i.e., that are outside of B, because this can lead to infinite loops.

In the deterministic case, checking for simulation corresponds to simply checking for the reachability of bad states in the composition $P||Q$. Bad states are those (p,q)

procedure *CheckSimulation3*
  Input: labeled transition systems P,Q
  Output: whether $p_0 \preceq q_0$

  $R := \emptyset$, $A := \emptyset$, $B := \emptyset$
  $W := P_0 \times Q_0$, $V = \emptyset$
  while $W \neq \emptyset$ **and** $V \neq \emptyset$ **do**
    while $W \neq \emptyset$ **and** $V = \emptyset$ **do**
      $A := A \cup W$
      $V := \{(p,q) \in W | p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p',q') \notin B\}$
      $W := post(W) \backslash A$
    end while
    while $V \neq \emptyset$ **do**
      $V := \{(p,q) \in (pre(V) \cap A) \backslash B | p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p',q') \notin B\}$
      $B := B \cup V$
    od
    if $\exists p_0 : \nexists q_0 : (p_0, q_0) \in B$ **then return** *false*
  od
  return *true*

Figure 3.7: On-the-fly algorithm for checking simulation of labeled transition systems

in which p has an outgoing transition with a label that is not in any outgoing transition of q.

Proposition 3.25. *If Q is deterministic, then $P \preceq Q$ if and only if for all (p,q) in $reach_{P||Q}$ holds that $p \preceq_1 q$.*

This reduces the complexity of checking for simulation of states tremendously, since the simulation relation cannot exist as soon as a bad state is found. The corresponding on-the-fly algorithm is shown in Fig. 3.8.

3.4 Related Work

Compositionality A more refined definition of compositionality is given by Zwiers [Zwi89], and it was put in the context of related work by de Roever in [Roe98]. The term *compositionality* describes a top-down perspective, in which for any global specification there exists a decomposed specification for each module that together allow the deduction of that global property. Expressed by simulation, this means that for any Q there exist Q_i with $P_i \preceq Q_i$ such that $Q_1 || \ldots || Q_n \preceq Q$. In our framework, this follows trivially since system and specification are given in the same formalism, as automata. A trivial choice is $Q_i \equiv P_i$. On the other hand, *modularity* refers to a bottom-up perspective, in which any global property that follows from the modular specifications must also be provable in the proof system at hand. So for any Q with $Q_1 || \ldots || Q_n \preceq Q$ it

procedure *CheckSimulation4*
Input: labeled transition systems P (possibly non-deterministic)
and deterministic Q
Output: whether $p_0 \preceq q_0$

$A := \emptyset$
$W := P_0 \times Q_0$
while $W \neq \emptyset$ **do**
$A := A \cup W$
$V := \{(p,q) \in W | p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q'\}$
if $V \neq \emptyset$ **then return** *false*
$W := post(W) \backslash A$
od
return *true*

Figure 3.8: On-the-fly algorithm for checking simulation with deterministic specification

must hold that $P_1 || \ldots || P_n \preceq Q$, which is fulfilled by our definition of compositionality and the transitivity of simulation.

Van der Schaft and Schumacher give a similar definition in [SS01] as *rule of modular behavior*. Alur et al. have presented such a rule for hierarchical hybrid systems based on trace inclusion in [AGLS01]. An extensive overview on simulation for discrete systems is given by Lynch and Vaandrager in [LV95]. Van Glabbeek provides in [vG01] an introduction and survey on the concept of branching time, and its implications on bisimulation equivalences. Algorithms for computing simulation relations for finite and infinite discrete systems can be found, e.g., in [HHK95]. An algorithm by Tan and Cleaveland [TC01] combines the benefits of two classic algorithms, the simulation algorithm by Bloom and Paige [BP95] and the bisimulation-minimization algorithm by Paige and Tarjan [PT87].

Simulation A framework for compositional reasoning based on a precongruence over Kripke structures was established by Grumberg and Long [GL91]. A version closer to simulation (but based on infinite paths) can be found in [Lon93], and a definition based on fair simulation is presented in [CGP99]. Compared to the structure in this thesis as shown in Fig. 3.1, Prop. 3.18 is established by constructing a witnessing simulation relation instead of being derived. The proof rule for decomposition is not explicitly derived in [GL91], but all its building blocks are present.

Simulation relations are frequently applied for verifying programs because they provide a sufficient condition for trace inclusion at a lower complexity. The notion of simulation used here is also referred to as *forward simulation*. For a sufficient and necessary criterion about trace inclusion, it is necessary to check forward as well as *backward simulation* [SAGG+93]. In [SAGG+93], simulation of two high-level spec-

ifications for a communication protocol was verified using the Larch Shared Language (LSL) and the Larch Proof Assistant (LP). They conclude:

> *We use simulation proofs because we believe that this method captures formally the natural structure of many informal correctness proofs for both finite and infinite state systems. In particular, it catches the structure of proofs based on successive refinements. [...]*
> *Simulation relations, like invariants, tend to capture central ideas; hence, they provide important documentation for algorithms. Simulations also tend to be readily modifiable when implementations are modified or when related algorithms are considered.*

This immediately relates to the context of reusable models in industrial applications and provides a strong argument for simulation relations, especially in the domain of embedded systems. Simulation between LTSs only preserves safety properties. Fair simulation, which is capable of checking liveness, has been examined, e.g., in [CGP99] and [HKR97]. *Tools* that support simulation for discrete automata and labeled transition systems are FC2TOOLS [BRRS96], Aldébaran [FKM93] and Mocha [HLQR99], see also p. 62.

Chapter 4

Assume-Guarantee Reasoning

Compositional verification as presented in the last chapter required a conservative abstraction for each component of the system. For a system $P = P_1||\ldots||P_n$, the construction of an abstracted version Q_i for each component P_i, i.e., $P_i \preceq Q_i$, the compositionality of simulation allows us to conclude that $P_1||\ldots||P_n \preceq Q_1||\ldots||Q_n$. However, in many cases a direct abstraction of the form $P_i \preceq Q_i$ yields little room for simplification, so one might not find Q_i with a substantially lower complexity than P_i. It is the interaction of the subsystems that produces the final behavior of the plant, which is only a small fragment of the product of all possible behaviors of each subsystem on its own. Therefore, a much simpler abstraction Q_i can be constructed if some *assumptions* about the behavior of the rest of the system are made. For example, a tank with inlet and outlet valves could be in one of four modes: the inlet or the outlet valve open, or both open or closed simultaneously. If the assumption can be made that inlet and outlet valves are never open at the same time, the model only has to reflect three modes. A *guarantee* of P_i is an aspect of its behavior that is true when the assumption is fulfilled.

There are two ways to combining assumptions and guarantees: In the first case, the assumption that justified the abstraction Q_i is independent of the guarantees of P_i, which is in the following referred to as *non-circular assume-guarantee reasoning*. Its soundness follows directly from compositionality. In the second case, the assumption directly or indirectly depends on the guarantees of P_i, which leads to a circular structure. This circularity requires additional conditions to ensure the conclusion is sound. In this chapter, we present such assume/guarantee conditions for simulation, and discuss how they can be computed compositionally. In the following, assume-guarantee reasoning is presented for a system $P_1||P_2$ to show that $P_1||P_2 \preceq Q_1||Q_2$. The generalization to an arbitrary number of subsystems is straightforward.

The next section recalls standard non-circular assume/guarantee reasoning for simulation, and in a more compact form for Σ-simulation, since the right hand side of the simulation inequality can be decomposed. In Sect. 4.2, we motivate the necessity of the assume/guarnatee conditions with some examples, and propose our assume/guarantee theorem for simulation. The Σ-simulation version, proposed in Sect. 4.3, is again

more compact, but the assume/guarantee conditions are more complex. A summary of related work can be found in Sect. 4.4.

4.1 Non-circular A/G-Reasoning

Non-circular A/G-Reasoning with Simulation A simple form of assume-guarantee reasoning relies on the fact that simulation is a precongruence. The abstraction of one automaton serves as the guarantee to another:

$$\frac{\begin{array}{rcl} P_1 & \preceq & Q_1 \\ Q_1||P_2 & \preceq & Q_1||Q_2 \end{array}}{P_1||P_2 \preceq Q_1||Q_2}. \tag{4.1}$$

This yields a triangular structure. The proof is straightforward: $P_1 \preceq Q_1 \Rightarrow P_1||P_2 \preceq Q_1||P_2$ due to invariance under composition. Through transitivity follows from $Q_1||P_2 \preceq Q_2$ that $P_1||P_2 \preceq Q_1||Q_2$. The following lemma allows a more efficient computation of $Q_1||P_2 \preceq Q_1||Q_2$:

Lemma 4.1. [1] *For any labeled transition systems P_1, Q_1 and Q_2 with $\Sigma_{P_1} = \Sigma_{Q_1}$, if a simulation relation R witnesses $Q_1||P_2 \preceq Q_1||Q_2$, then so does*

$$R' = \{((q_1,p_2),(q_1,q_2)) | \exists \hat{q}_1 : ((q_1,p_2),(\hat{q}_1,q_2)) \in R\}. \tag{4.2}$$

Proof. First we show that R' is a simulation relation, and afterwards that the initial states are properly contained in R'. Consider a state $((q_1,p_2),(q_1,q_2)) \in R'$ with a transition $(q_1,p_2) \xrightarrow{\alpha} (q_1',p_2')$. We will show that this implies a transition $(q_1,q_2) \xrightarrow{\alpha} (q_1',q_2')$ with $((q_1',p_2'),(q_1',q_2') \in R'$. By definition of R', there exists a $\hat{q}_1$ such that

$$((q_1,p_2),(\hat{q}_1,q_2)) \in R. \tag{4.3}$$

- If $\alpha \in \Sigma_{Q_1} \cap \Sigma_{P_2}$, this implies transitions $q_1 \xrightarrow{\alpha}_{Q_1} q_1'$ and $p_2 \xrightarrow{\alpha} p_2'$. Because (4.3), there exists a transition $(\hat{q}_1,q_2) \xrightarrow{\alpha} (\hat{q}_1',q_2')$ with $((q_1',p_2'),(\hat{q}_1',q_2')) \in R$. With $\alpha \in \Sigma_{Q_1} \cap \Sigma_{P_2}$ this implies a transition $q_2 \xrightarrow{\alpha} q_2'$, so that in state (q_1,q_2) there is also a transition $(q_1,q_2) \xrightarrow{\alpha} (q_1',q_2')$, and with $((q_1',p_2'),(\hat{q}_1',q_2')) \in R$ follows that $((q_1',p_2'),(q_1',q_2')) \in R'$.

- If $\alpha \in \Sigma_{Q_1} \setminus \Sigma_{P_2}$, this implies a transition $q_1 \xrightarrow{\alpha}_{Q_1} q_1'$ and $p_2 = p_2'$, and by the definition of parallel composition also a transition $(q_1,q_2) \xrightarrow{\alpha} (q_1',q_2)$. Because of (4.3), there exists a transition $(\hat{q}_1,q_2) \xrightarrow{\alpha} (\hat{q}_1',q_2)$ with $((q_1',p_2),(\hat{q}_1',q_2)) \in R$, so that $((q_1',p_2),(q_1',q_2)) \in R'$.

[1] This lemma was contributed by Frits Vaandrager, whose authorship is gratefully acknowledged.

- If $\alpha \in \Sigma_{P_2} \setminus \Sigma_{Q_1}$, there is a transition $p_2 \xrightarrow{\alpha}_{P_2} p_2'$ and $q_1 = q_1'$. Because of (4.3), there exists a transition $(\hat{q}_1, q_2) \xrightarrow{\alpha} (\hat{q}_1, q_2')$ with $((q_1, p_2'), (\hat{q}_1, q_2')) \in R$. By definition of parallel composition, this implies a transition $q_2 \xrightarrow{\alpha}_{Q_2} q_2'$, and consequently a transition $(q_1, q_2) \xrightarrow{\alpha} (q_1, q_2')$. Since $((q_1, p_2), (\hat{q}_1, q_2')) \in R$, it follows that $((q_1, p_2), (q_1, q_2')) \in R'$.

Since R is a witness, it holds that for any $(q_1, p_2) \in Q_{01} \times P_{02}$ there exists a pair $(\hat{q}_1, q_2) \in Q_{01} \times Q_{02}$ such that $((q_1, p_2), (\hat{q}_1, q_2)) \in R$. Since $(q_1, q_2) \in Q_{01} \times Q_{02}$, and $((q_1, p_2), (q_1, q_2)) \in R'$, it follows that for any state $(q_1, p_2) \in Q_{01} \times P_{02}$ there exists also a corresponding state in R'. □

Lemma 4.1 allows to compute R over the state space $S_{Q_1} \times S_{P_2} \times S_{Q_2}$ instead of $S_{Q_1} \times S_{P_2} \times S_{Q_1} \times S_{Q_2}$. With Corollary 3.21, the inequality $Q_1||P_2 \preceq Q_1||Q_2$ can also be simplified to $Q_1||P_2 \preceq Chaos(Q_1)||Q_2$.

Non-circular A/G-Reasoning with Σ-Simulation Thanks to the decomposition of the specification, the proof rule is simpler for Σ-simulation:

$$\frac{\begin{array}{rcl} P_1 & \preceq & Q_1 \\ Q_1||P_2 & \preceq & Q_2 \end{array}}{P_1||P_2 \preceq Q_1||Q_2}. \tag{4.4}$$

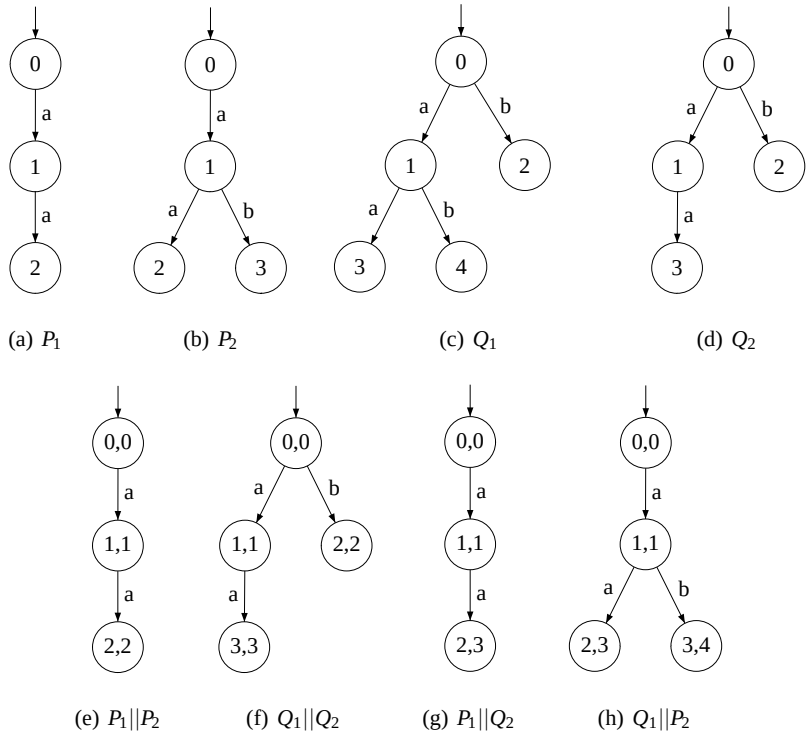
$P_1 \preceq Q_1 \Rightarrow P_1||P_2 \preceq Q_1||P_2$ due to invariance under composition. Also, $P_1||P_2 \preceq Q_1||P_2 \Leftrightarrow P_1||P_2 \preceq Q_1$ by decomposition of the specification according to Theorem 3.20. Through transitivity follows from $Q_1||P_2 \preceq Q_2$ that $P_1||P_2 \preceq Q_2$. By composition of the specification, also Theorem 3.20, follows that $P_1||P_2 \preceq Q_1||Q_2$.

4.2 Circular A/G-Reasoning with Simulation

In circular assume-guarantee reasoning the components P_1 and P_2 interact so closely that neither one fulfills Q_1 or Q_2 directly, i.e., $P_1 \not\preceq Q_1$ and $P_2 \not\preceq Q_2$. To restrict the behavior of each module, P_1 is composed with Q_2 and P_2 with Q_1. However, such a proof is only sound if additional conditions ensure that Q_1 and Q_2 do not block transitions that are enabled in the composition of P_1 and P_2. It is known that these conditions must be non-propositional, e.g., use induction [VV01, HQRT02]. The basic structure is:

$$\frac{\begin{array}{c} P_1||Q_2 \preceq Q_1||Q_2 \\ Q_1||P_2 \preceq Q_1||Q_2 \\ \textit{A/G-conditions} \end{array}}{P_1||P_2 \preceq Q_1||Q_2}. \tag{4.5}$$

Three fundamental properties shall be illustrated by examples:



(a) P_1 (b) P_2 (c) Q_1 (d) Q_2

(e) $P_1||P_2$ (f) $Q_1||Q_2$ (g) $P_1||Q_2$ (h) $Q_1||P_2$

Figure 4.1: A/G-reasoning yields a false negative for Ex. 4.1

- The rule is not complete, even without A/G-conditions.
- The rule is not sound without appropriate A/G-conditions.
- The A/G conditions must be at least as expressive as simulation.

The first example shows that the proof rule is not complete, i.e. there are systems that fulfill $P_1||P_2 \preceq Q_1||Q_2$, but not $P_1||Q_2 \preceq Q_1||Q_2$ and $Q_1||P_2 \preceq Q_1||Q_2$:

Example 4.1. *Figure 4.1 shows automata P_1, P_2, Q_1 and Q_2, all with alphabets $\Sigma_{P_1} = \Sigma_{P_2} = \Sigma_{Q_1} = \Sigma_{Q_2} = \{a, b\}$, and the composed automata $P_1||P_2$, $Q_1||Q_2$, $P_1||Q_2$ and $Q_1||P_2$ that arise in circular A/G-reasoning. Even though $P_1||P_2 \preceq Q_1||Q_2$, circular A/G-reasoning fails because $Q_1||P_2 \not\preceq Q_1||Q_2$. Note that non-circular A/G-reasoning also isn't possible: $P_1 \preceq Q_1$, but $Q_1||P_2 \not\preceq Q_1||Q_2$.*

Without the A/G-conditions, the rule is not sound. It is entirely up to the A/G-conditions to ensure that violations as in the following example are detected:

Example 4.2. *Consider a system, where P_1 and P_2 are chaos, and neither specification Q_1 nor Q_2 have any transitions, formally:*

$$\begin{aligned} P_1 &= Chaos(\Sigma), \quad Q_1 = (Q_{01}, \Sigma, \emptyset, Q_{01}), \\ P_2 &= Chaos(\Sigma), \quad Q_2 = (Q_{02}, \Sigma, \emptyset, Q_{02}). \end{aligned}$$

Then the parallel composition $P_1||Q_2 = (P_{01} \times Q_{02}, \Sigma, \emptyset, P_{01} \times Q_{02})$ has no transitions, and neither do $Q_1||P_2$ and $Q_1||Q_2$. Therefore both $P_1||Q_2 \preceq Q_1||Q_2$ and $Q_1||P_2 \preceq Q_1||Q_2$ hold. Yet $P_1||P_2 \cong Chaos(\Sigma)$, so that the system does not fulfill the specification:

$$P_1||P_2 \not\preceq Q_1||Q_2.$$

It is important to note that the A/G-conditions are not just a simple condition for ensuring soundness. As the next example illustrates, they are a vital element to the meaning of such a proof. The A/G-conditions must be as least as expressive as simulation, and they must differentiate between the system and the specification, i.e., what is on the left hand side of the conclusion and what is on the right hand side.

Example 4.3. *Consider a LTSs P_2, P_3, Q_1, Q_2, Q_3 with equal alphabets*[2]*, and the following two proof rules:*

$$\text{(a)}\quad \frac{\begin{array}{c} P_2||(Q_1||Q_3) \preceq Q_1||Q_2 \\ (Q_1||Q_2)||P_3 \preceq Q_1||Q_3 \\ \textit{A/G-conditions} \end{array}}{P_2||P_3 \preceq Q_1||Q_2||Q_3} \qquad \text{(b)}\quad \frac{\begin{array}{c} (Q_1||P_2)||Q_3 \preceq Q_2 \\ Q_2||(Q_1||P_3) \preceq Q_3 \\ \textit{A/G-conditions} \end{array}}{Q_1||P_2||P_3 \preceq Q_2||Q_3}.$$

The upper inequalities are equivalent for both proofs:

$$\begin{aligned} P_2||(Q_1||Q_3) \preceq Q_1||Q_2 &\Leftrightarrow P_2||(Q_1||Q_3) \preceq Q_1 \wedge P_2||(Q_1||Q_3) \preceq Q_2 \\ &\Leftrightarrow P_2||(Q_1||Q_3) \preceq Q_2 \Leftrightarrow (Q_1||P_2)||Q_3 \preceq Q_2, \end{aligned}$$

and similarly for the second inequality. Yet the conclusions are entirely different. Consider that the conclusion of (a) implies $P_2||P_3 \preceq Q_1$, while in (b) Q_1 relaxes the specification. The consequences are obvious if one considers a Q_1 without any transitions. While the conclusion of (a) is only fulfilled by a system $P_2||P_3$ without transitions, the conclusion of (b) would always be fulfilled (as long as $Q_2||Q_3$ has an initial state).

The examples showed that there can be transitions in the composed system $P_1||P_2$ that can be impossible to check for validity by simply looking at the inequalities in (4.5). While the desired conclusion $P_1||P_2 \preceq Q_1||Q_2$ is a statement over a subset of the state space of all four automata P_1, P_2, Q_1 and Q_2, the inequalities $P_i||Q_j \preceq Q_1||Q_2$ only refer to states in the subspace over the three automata P_i, Q_i and Q_j. One could say that common transitions of $P_1||P_2$, i.e., those with a label in both $\Sigma_{P_1} and \Sigma_{P_2}$ are not predictable in those subspaces. They must be accounted for by limiting the transitions that are allowed to occur in (4.5). This is formalized by the following Theorem:

[2]The same argument can be made for arbitrary alphabets using Σ-simulation.

Theorem 4.2 (A/G-simulation)**.** *Consider LTSs P_1, P_2, Q_1 and Q_2 with alphabets $\Sigma_{P_1} = \Sigma_{Q_1}$, $\Sigma_{P_2} = \Sigma_{Q_2}$ for which simulation relations R_1 and R_2 witness, respectively,*

$$P_1 \,||\, Q_2 \;\preceq\; Q_1 \,||\, Q_2 \quad (R_1) \text{ and} \tag{4.6}$$
$$Q_1 \,||\, P_2 \;\preceq\; Q_1 \,||\, Q_2. \quad (R_2) \tag{4.7}$$

Let the relations R_1' and R_2' be defined by

$$R_1' \;=\; \{(p_1, q_1, q_2) | \exists \hat{q}_2 : ((p_1, q_2), (q_1, \hat{q}_2)) \in R_1\}, \tag{4.8}$$
$$R_2' \;=\; \{(p_2, q_1, q_2) | \exists \hat{q}_1 : ((q_1, p_2), (\hat{q}_1, q_2)) \in R_2\}. \tag{4.9}$$

Then the relation

$$R = \{((p_1, p_2), (q_1, q_2)) | (p_1, q_1, q_2) \in R_1' \wedge (p_2, q_1, q_2) \in R_2'\} \tag{4.10}$$

is a simulation relation [3] *for $P_1 \,||\, P_2 \preceq Q_1 \,||\, Q_2$ if and only if for all $((p_1, p_2), (q_1, q_2)) \in R$ and $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$ the following* **A/G-condition** *holds:*

$$(p_1, p_2) \xrightarrow{\alpha}_{P_1 || P_2} (p_1', p_2') \Rightarrow (\exists q_1' : q_1 \xrightarrow{\alpha}_{Q_1} q_1' \vee \exists q_2' : q_2 \xrightarrow{\alpha}_{Q_2} q_2'). \tag{4.11}$$

Proof. Consider the relation

$$R = \{((p_1, p_2), (q_1, q_2)) | \exists \hat{q}_2 : ((p_1, q_2), (q_1, \hat{q}_2)) \in R_1 \wedge \exists \hat{q}_1 : ((q_1, p_2), (\hat{q}_1, q_2)) \in R_2\}.$$

We show that under the hypothesis a transition $(p_1, p_2) \xrightarrow{\alpha}_{P_1 || P_2} (p_1', p_2')$ implies a transition $(q_1, q_2) \xrightarrow{\alpha}_{Q_1 || Q_2} (q_1', q_2')$ such that $((p_1', p_2'), (q_1', q_2')) \in R$. Consider such a transition for $(p_1, p_2, q_1, q_2) \in R$, so that there exist $\hat{q}_1, \hat{q}_2$ with $((p_1, q_2), (q_1, \hat{q}_2)) \in R_1$ and $((q_1, p_2), (\hat{q}_1, q_2)) \in R_2$. Let $\Sigma_1 = \Sigma_{P_1} = \Sigma_{Q_1}$ and $\Sigma_2 = \Sigma_{P_2} = \Sigma_{Q_2}$.

(a) If $\alpha \in \Sigma_1 \backslash \Sigma_2$, then by the definition of parallel composition there is a transition

$$p_1 \xrightarrow{\alpha}_{P_1} p_1',$$

and P_2 does not participate in the transition, i.e., $p_2 = p_2'$. Since $\alpha \notin \Sigma_2$, it follows by the definition of parallel composition that there is a transition

$$(p_1, \hat{q}_2) \xrightarrow{\alpha}_{P_1 || Q_2} (p_1', \hat{q}_2).$$

Since $((p_1, q_2), (q_1, \hat{q}_2)) \in R_1$ this implies that there is some q_1' with a transition

$$(q_1, \hat{q}_2) \xrightarrow{\alpha}_{Q_1 || Q_2} (q_1', \hat{q}_2)$$

[3] Note that R might not be a witness for $P_1 || P_2 \preceq Q_1 \,||\, Q_2$, since not all initial states of $P_1 || P_2$ are guaranteed to have a matching initial state in $Q_1 || Q_2$.

and therefore a transition

$$q_1 \xrightarrow{\alpha}_{Q1} q_1'$$

exists with $((p_1', q_2), (q_1', \hat{q}_2)) \in R_1$. With $((q_1, p_2), (\hat{q}_1, q_2)) \in R_2$, it follows that there is a transition

$$(q_1, p_2) \xrightarrow{\alpha}_{Q_1||P_2} (q_1', p_2),$$

which in turn implies that there exists a $\overline{q}_1$ with a transition

$$(q_1, \hat{q}_2) \xrightarrow{\alpha}_{Q_1||Q_2} (\overline{q}_1', \hat{q}_2)$$

and $((q_1', p_2), (\overline{q}_1', q_2)) \in R_2$, and therefore $((p_1', p_2), (q_1', q_2)) \in R$.

(b) If $\alpha \in \Sigma_2 \backslash \Sigma_1$, a symmetric argument to (a) proves that a transition

$$(q_1, q_2) \xrightarrow{\alpha}_{Q_1||Q_2} (q_1, q_2')$$

exists with $((p_1', p_2'), (q_1', q_2')) \in R$.

(c) Otherwise $\alpha \in \Sigma_1 \cap \Sigma_2$ and there is a transition

$$p_1 \xrightarrow{\alpha}_{P_1} p_1' \text{ as well as } p_2 \xrightarrow{\alpha}_{P_2} p_2'.$$

According to the A/G-condition of Theorem 4.2 there is a transition in Q_1 or in Q_2. Let's consider a transition in Q_2, a transition in Q_1 is symmetrical. From $((p_1, q_2), (q_1, \hat{q}_2)) \in R_1$ follows that if

$$q_2 \xrightarrow{\alpha}_{Q_2} q_2' \tag{4.12}$$

then there exists a $\hat{q}_2'$ and a transition

$$(q_1, q_2) \xrightarrow{\alpha}_{Q_1||Q_2} (q_1', \hat{q}_2').$$

This implies a transition

$$q_1 \xrightarrow{\alpha}_{Q_1} q_1'$$

and that $((p_1', q_2'), (q_1', \hat{q}_2')) \in R_1$. We must now show that $(p_2', q_1', q_2') \in R_2$. Since $(p_2, q_1, q_2) \in R_2'$ there exists some $\hat{q}_1$ with $((q_1, p_2), (\hat{q}_1, q_2)) \in R_2$. This implies that there is a transition

$$(q_1, p_2) \xrightarrow{\alpha}_{Q_1||P_2} (q_1', p_2'),$$

which in turn implies there exist $\hat{q}_1', \overline{q}_2'$ and a transition

$$(\hat{q}_1, q_2) \xrightarrow{\alpha}_{Q_1||Q_2} (\hat{q}_1', \overline{q}_2')$$

with $((q_1', p_2'), (\hat{q}_1', \overline{q}_2')) \in R_2$. Since the initial choice of q_2' in (4.12) was not restricted, we can set $q_2' = \overline{q}_2'$, and consequently $((p_1', p_2'), (q_1', q_2')) \in R$, which shows that R is a simulation relation.

The necessary direction of Theorem 4.2 follows directly from the definitions of simulation and parallel composition. For all $((p_1, p_2), (q_1, q_2))$ in any simulation relation for $P_1||P_2 \preceq Q_1||Q_2$, and any label $\alpha \in \Sigma_{P_1} \cup \Sigma_{P_2}$ it holds that a transition $(p_1, p_2) \xrightarrow{\alpha}_{P_1||P_2} (p_1', p_2')$ implies either

- $\alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2}$: $\exists q_1', q_2' : q_1 \xrightarrow{\alpha}_{Q_1} q_1' \wedge q_2 \xrightarrow{\alpha}_{Q_2} q_2'$, or
- $\alpha \in \Sigma_{Q_i} \cap \Sigma_{Q_j}$, $(i, j) \in \{(1,2), (2,1)\}$: $\exists q_i' : q_i \xrightarrow{\alpha}_{Q_i} q_i'$.

It is easy to see that the A/G-condition is fulfilled in any of these cases. □

Note that condition (4.11) is maximal in the sense that it is fulfilled for any relation that witnesses $P_1||P_2 \preceq Q_1||Q_2$. This is underlined by the following corollary, which relates the A/G-conditions to $\preceq_1$-simulation (see Def. 3.22, p. 33):

Corollary 4.3. *Given that $P_1||Q_2 \preceq Q_1||Q_2$ and $Q_1||P_2 \preceq Q_1||Q_2$, it holds that a relation R according to (4.10) is a simulation relation for $P_1||P_2 \preceq Q_1||Q_2$ if and only if it is a simulation relation for $P_1||P_2 \preceq_1 Q_1||Q_2$.*

Proof. For the sufficient direction, it follows directly from the definition of $\preceq_1$-simulation that the A/G-condition is fulfilled. The necessary direction is guaranteed by Prop. 3.23. □

Assume/guarantee-reasoning based on Theorem 4.2 is carried out in the following steps:

1. Initialize R_1 and R_2, e.g., with the entire or only the reachable state space.
2. Construct R from R_1 and R_2.
3. Trim states in R that violate the A/G-condition (4.11).
4. If states were trimmed, turn R into a simulation relation, e.g., with a fixed-point algorithm.
5. If the initial states are contained in R, then $P_1 \,||\, P_2 \preceq Q_1 \,||\, Q_2$ holds.

This approach is not compositional, because R is constructed for the composed system and subject to a fixed-point computation in step 4. It yields an advantage over a non-compositional check if the fixed-point computation of R_1 and R_2 is significantly faster than computing R in the first place. This is most likely the case if few states need to be trimmed from R.

If possible, we would like to avoid the explicit construction of R and computations in the space of $P_1||P_2$ as much as possible. We can do so under the premise that the initial states are only a small part of the entire state space. We preemptively remove potentially violating states already from R_1 and R_2, so that we are guaranteed that R fulfills the A/G-condition (4.11). The check of the initial states remains, but does not necessarily require to construct all of R. In the procedure above, we replace steps 2.–4. with:

2^*. Trim states in R_1 and R_2 that could violate the A/G-condition (4.11) in R.

3^*. Turn R_1 and R_2 into simulation relations.

4^*. Construct R only for the initial states.

Sufficient conditions are used to perform step 2^*. The following corollary applies to a number of practical cases, and follows immediately from Theorem 4.2 and the definition of non-blocking:

Corollary 4.4. *A/G-reasoning according to (4.5) is sound if Q_1 is non-blocking over Σ_1 and Q_2 is non-blocking over Σ_2 with $\Sigma_1 \cup \Sigma_2 = \Sigma_{Q_1} \cap \Sigma_{Q_2}$.*

It is often demanded in assume-guarantee reasoning that Q_1 and Q_2 are non-blocking, see e.g. [HQRT02], and it is easy to check for finite systems.[4] In the next couple of sections we present other sufficient criteria for step 2^*.

4.2.1 Separate Trimming

Ideally, one would like to trim R_1 and R_2 separately, i.e., without taking into account the pairs of states that eventually constitute R in (4.10). In such a scheme, a state $((p_1, q_2), (q_1, \cdot))$ is removed from R_1 if

$$\exists \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2}, p_1' \in S_{P_1} : p_1 \xrightarrow{\alpha} p_1' \wedge \nexists q_1' : (q_1 \xrightarrow{\alpha} q_1') \wedge \nexists q_2' : (q_2 \xrightarrow{\alpha} q_2'). \quad (4.13)$$

The computation of R_2 is performed symmetrically to R_1. The overall algorithm for obtaining R is shown in Fig. 4.2. It follows directly from the definition of parallel composition that it guarantees the A/G-condition (4.11). In it, R_1 and R_2 are initialized with the set of reachable states and turned into simulation relations before the trimming so as few states as possible are trimmed because of condition (4.13).

Example 4.4. *Consider the automata shown in Fig. 4.3. Both $P_1||Q_1 \preceq Q_1||Q_2$ and $P_2||Q_2 \preceq Q_1||Q_2$ hold. The simulation relations R_1 for $P_1||Q_1 \preceq Q_1||Q_2$ and R_2 for $Q_1||P_2 \preceq Q_1||Q_2$ are shown in Tab. 4.1. All the states fulfill condition (4.13), so that none have to be trimmed.*

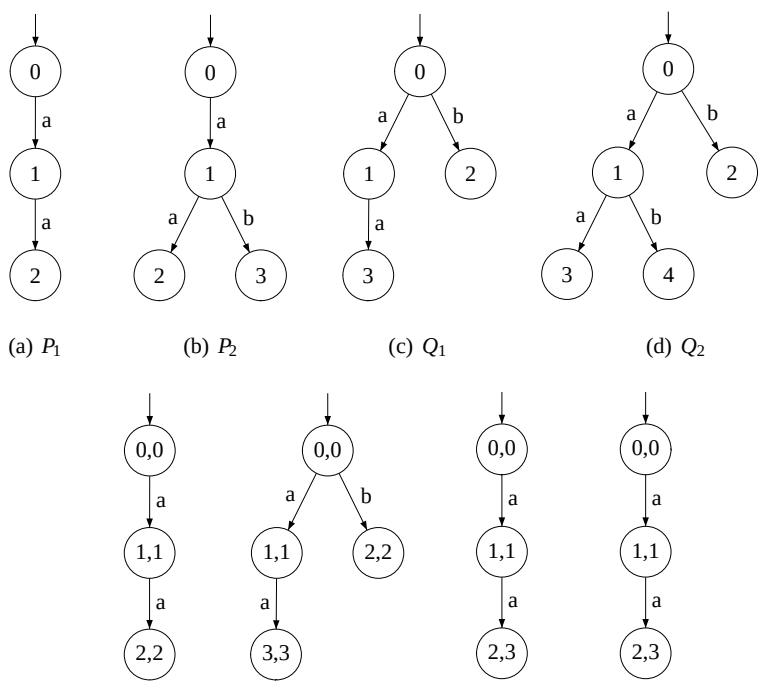
Unfortunately, separate trimming is unsuccessful in many, if not most, practical cases. This forces us to consider schemes that relate R_1 and R_2, such as the one presented in the next section.

[4] Corollary 4.4 is consistent with the assume/guarantee-rule for simulation by Henzinger et al. in [HQRT02]. There, the A/G-rule for Moore machines is shown to be sound if all automata are non-blocking in the input labels and there is finite non-determinism. Since input- and output labels are disjoint, and the output labels of the Moore machines that are composed are disjoint, In the corresponding setting using LTSs, the conditions of Corollary 4.4 are fulfilled, so that our rule assures soundness as well.

procedure *CheckAGSimulationSeparate*
Input: labeled transition systems P_1, P_2, Q_1, Q_2
Output: A/G-simulation relations R_1, R_2

$R_1 := reach_{P_1||Q_2||Q_1||Q_2}$, $R_2 := reach_{Q_1||P_2||Q_1||Q_2}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1||Q_2}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_1||Q_2}(R_2)$
$R_1 := R_1 \setminus \{(p_1, q_2, q_1, \cdot) | \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_1' : p_1 \xrightarrow{\alpha} p_1' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}$
$R_2 := R_2 \setminus \{(q_1, p_2, \cdot, q_2) | \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_2' : p_2 \xrightarrow{\alpha} p_2' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1||Q_2}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_1||Q_2}(R_2)$

Figure 4.2: Algorithm for A/G-simulation by separate trimming



(a) P_1 (b) P_2 (c) Q_1 (d) Q_2

(e) $P_1||P_2$ (f) $Q_1||Q_2$ (g) $P_1||Q_2$ (h) $Q_1||P_2$

Figure 4.3: A/G-reasoning with separate trimming works for Ex. 4.4

Table 4.1: Simulation relations for Ex. 4.4 after separate trimming

(a) R_1

$P_1 \| Q_2 \backslash^{Q_1 \| Q_2}$	(0,0)	(1,1)	(2,2)	(3,3)
(0,0)	•	–	–	–
(1,1)	•	•	–	–
(2,3)	•	•	•	•

(b) R_2

$Q_1 \| P_2 \backslash^{Q_1 \| Q_2}$	(0,0)	(1,1)	(2,2)	(3,3)
(0,0)	•	–	–	–
(1,1)	•	•	–	–
(2,3)	•	•	•	•

4.2.2 Composite Trimming

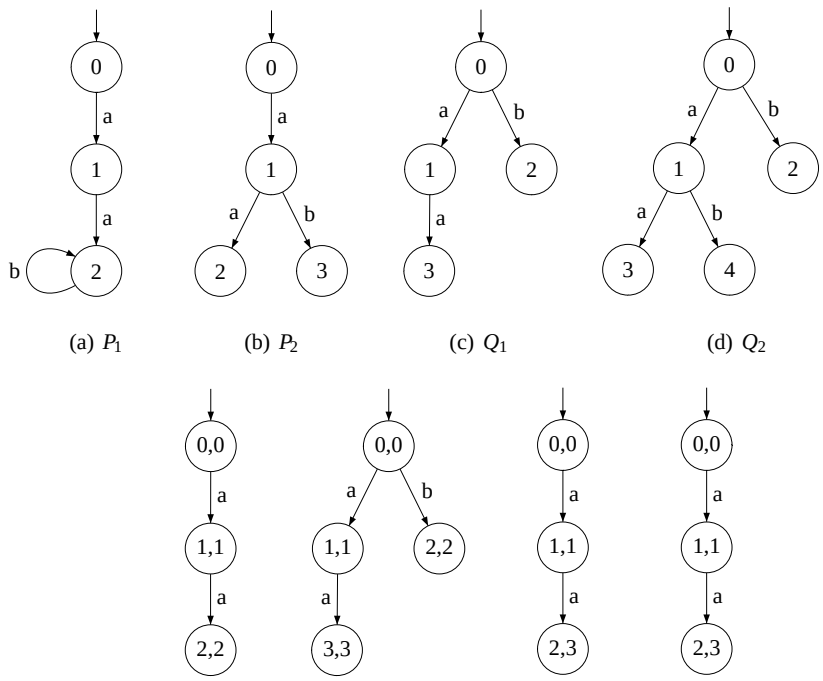
In many cases the interaction between the modules is such that a separate trimming is too restrictive and R_1 and R_2 must be considered together. Informally, this can be explained as follows: A safety specification basically consists of forbidding undesired actions. If these are actions common to P_1 and P_2, they are either forbidden (don't occur) in Q_1 and Q_2 or they disappear in the composition of $Q_1||Q_2$. In the former case, P_1 has transitions that are neither in Q_1 nor Q_2, and a symmetric argument holds for P_2. Whether those transitions are still in $P_1||P_2$ can not be decided by simply looking at $P_1|||Q_2$ or $Q_1||P_2$. But there is a sufficient condition: If for all states $(\cdot, q_2, q_1) \in R_1'$ in which P_1 has such bad actions, P_2 doesn't, i.e. $(q_1, p_2, q_2) \in R_2'$ implies that there is no outgoing transition with the bad label.

Example 4.5. *Consider the automata shown in Fig. 4.4, where P_1 of Ex. 4.4 was fitted with a self-loop with label b in state $p_1 = 2$. The composed automata are the same as those shown in Fig. 4.3(e)–(h). Also, the simulation relations are identical to the ones shown in Tab. 4.1, so that $P_1 \,||\, Q_2 \preceq Q_1 \,||\, Q_2$ and $Q_1 \,||\, P_2 \preceq Q_1 \,||\, Q_2$ both hold. However, the outgoing transition in state $p_1 = 2$ has no match in the associated states (q_1, q_2) in R_1. Trimming the states separately yields the relations R_1^* and R_2^* shown in Table 4.2. Iterating the fixed-point operator for simulation on R_1^* yields an empty relation, so that the proof is no longer possible. In order to be sound, it must be checked whether for any state $(p_1 = 2, p_2)$ in $P_1||P_2$ there is also an outgoing transition in P_2. In R_1, $p_1 = 2$ is associated with $(q_1, q_2) = (3,3)$. In R_2, $(q_1, q_2) = (3,3)$ is only associated with $(p_2, q_2) = (2,3)$. Since $p_2 = 2$ has no outgoing transition with label b, the conditions of Theorem 4.2 are satisfied.*

The sets of critical labels and states in R_1 and R_2 that could violate the A/G-conditions are:

$$D_{P_1} = \{(q_1, q_2, \alpha) | \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_1 : (p_1, q_2, q_1) \in R_1' \wedge \\ p_1 \xrightarrow{\alpha} p_1' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\} \quad (4.14)$$

$$D_{P_2} = \{(q_1, q_2, \alpha) | \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_2 : (q_1, p_2, q_2) \in R_2' \wedge \\ p_2 \xrightarrow{\alpha} p_2' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}. \quad (4.15)$$



(a) P_1 (b) P_2 (c) Q_1 (d) Q_2

(e) $P_1||P_2$ (f) $Q_1||Q_2$ (g) $P_1||Q_2$ (h) $Q_1||P_2$

Figure 4.4: A/G-reasoning with separate trimming fails, but composite trimming works for Ex. 4.5

Table 4.2: Simulation relations for Ex. 4.5 after separate trimming

(a) R_1^*

$P_1\|\|Q_2$ \ $Q_1\|\|Q_2$	(0,0)	(1,1)	(2,2)	(3,3)
(0,0)	•	–	–	–
(1,1)	•	•	–	–
(2,3)	–	–	–	–

(b) R_2^*

$Q_1\|\|P_2$ \ $Q_1\|\|Q_2$	(0,0)	(1,1)	(2,2)	(3,3)
(0,0)	•	–	–	–
(1,1)	•	•	–	–
(2,3)	•	•	•	•

procedure *CheckAGSimulationComposite*
Input: labeled transition systems P_1, P_2, Q_1, Q_2
Output: A/G-simulation relations R_1, R_2
$R_1 := reach_{P_1||Q_2||Q_1||Q_2}$, $R_2 := reach_{Q_1||P_2||Q_1||Q_2}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1||Q_2}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_1||Q_2}(R_2)$
$D_{P_1} = \{(q_1,q_2,\alpha)|\alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_1,\hat{q}_2 : (p_1,q_2,q_1,\hat{q}_2) \in R_1 \wedge$
$p_1 \xrightarrow{\alpha} p_1' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}$
$D_{P_2} = \{(q_1,q_2,\alpha)|\alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_2,\hat{q}_1 : (q_1,p_2,\hat{q}_1,q_2) \in R_2 \wedge$
$p_2 \xrightarrow{\alpha} p_2' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}$.
if $D_{P_1} \cap D_{P_2} \neq \emptyset$
$R_1 := R_1 \backslash \{(p_1,q_2,q_1,\cdot)|\exists p_1' : p_1 \xrightarrow{\alpha} p_1' \wedge (q_1,q_2,\alpha) \in D_{P_2}\}$
$R_2 := R_2 \backslash \{(q_1,p_2,\cdot,q_2)|\exists p_2' : p_2 \xrightarrow{\alpha} p_2' \wedge (q_1,q_2,\alpha) \in D_{P_1}\}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1||Q_2}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_1||Q_2}(R_2)$
end if

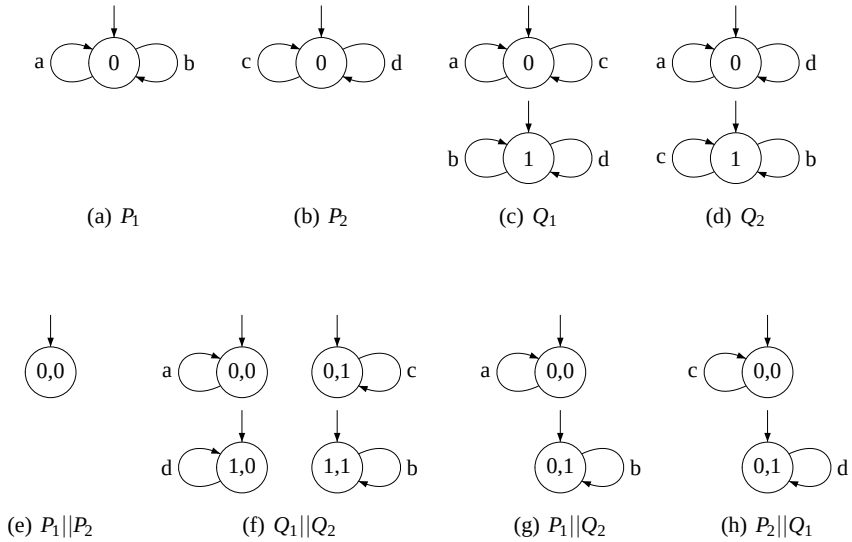
Figure 4.5: Algorithm for A/G-simulation by composite trimming

It is a sufficient condition for A/G-simulation that

$$D = D_{P_1} \cap D_{P_2} = \emptyset. \tag{4.16}$$

If there are such violating states, one can trim the relations. An element (p_1,q_2,q_1) of R_1 is removed if $p_1 \xrightarrow{\alpha} p_1' \wedge (q_1,q_2,\alpha) \in D_{P_2}$, and symmetrically for R_2. The proof concludes if there is a simulation relation R_1' that is a subset of the trimmed relation R_1. An algorithm is shown in Fig. 4.5, and using the definition of parallel composition it is easy to see that the resulting R_1 and R_2 fulfill the A/G-condition (4.11).

Note that if Q_1 and Q_2 are non-deterministic, there is a choice whether to remove a state in $D_{P_1} \cap D_{P_2}$ from R_1 or R_2. If Q_1 is deterministic, the bad state should be removed from R_2, and symmetrically for Q_2 and R_1. If both Q_1 and Q_2 are deterministic, there is no solution as soon as $D_{P_1} \cap D_{P_2}$ is non-empty. There is the possibility of trying all combinations, but that will likely defy the advantages of the A/G-approach compared to simply a composed analysis.



(a) P_1 (b) P_2 (c) Q_1 (d) Q_2

(e) $P_1||P_2$ (f) $Q_1||Q_2$ (g) $P_1||Q_2$ (h) $P_2||Q_1$

Figure 4.6: A/G-reasoning fails in Ex. 4.6 because of the initial states

4.2.3 Checking the Initial States

To finalize the A/G-proof, it must be shown that all the initial states of $P_1||P_2$ have a matching initial state of $Q_1||Q_2$ in R. For the simulation relation (4.10), it must be shown that for all $(p_1, p_2) \in P_{01} \times P_{02}$ there exist $(q_1, q_2) \in Q_{01} \times Q_{02}$ such that $(p_i, q_1, q_2) \in R_i$ for $i = 1, 2$. It follows from $P_i||Q_j \preceq Q_1||Q_2$ that for any p_i there exists some pair $(q_{1i}, q_{2i}) \in Q_{01} \times Q_{02}$, but the important point is that this must be the same pair for both p_1 and p_2.

Example 4.6. *Consider the automata shown in Fig. 4.6, all with the alphabet $\Sigma = \{a, b, c, d\}$. All states are initial states. Table 4.3 shows the A/G-relations R_1 for $P_1||Q_2 \preceq Q_1||Q_2$ and R_2 for $P_2||Q_1 \preceq Q_1||Q_2$ as well as their quantified counterparts R'_1 and R'_2. The simulation relation R constructed according to (4.10) is empty, while there exists a simulation relation R' that contains all states, since $P_1||P_2$ doesn't have any transitions. Therefore the A/G-reasoning fails in this example.*

It follows that the circular A/G-reasoning is only a sufficient condition with respect to the containment of the initial states. There exist cases in which R_1 and R_2 exist, but no simulation relation can be constructed from R_1 and R_2 that contains the initial states appropriately, even though a global R' exists and $P_1||P_2 \preceq Q_1||Q_2$ holds. A sufficient condition for the containment is that for all $(p_1, q_2) \in P_{01} \times Q_{02}$ and $q_1 \in Q_{01}$ holds $(p_1, q_1, q_2) \in R'_1$. Alternatively a symmetric argument is valid for R'_2.

Table 4.3: Simulation relations for Ex. 4.6 after separate trimming

(a) R_1

$P_1 \| Q_2 \backslash Q_1 \| Q_2$	(0,0)	(1,0)	(0,1)	(1,1)
(0,0)	•	–	–	–
(0,1)	–	–	–	•

(b) R_2

$P_2 \| Q_1 \backslash Q_1 \| Q_2$	(0,0)	(1,0)	(0,1)	(1,1)
(0,0)	–	–	•	–
(0,1)	–	•	–	–

(c) R_1'

$P_1 \backslash Q_1 \| Q_2$	(0,0)	(1,0)	(0,1)	(1,1)
0	•	–	–	•

(d) R_2'

$P_2 \backslash Q_1 \| Q_2$	(0,0)	(1,0)	(0,1)	(1,1)
0	–	•	•	–

(e) R

$P_1 \| P_2 \backslash Q_1 \| Q_2$	(0,0)	(1,0)	(0,1)	(1,1)
(0,0)	–	–	–	–

(f) R'

$P_1 \| P_2 \backslash Q_1 \| Q_2$	(0,0)	(1,0)	(0,1)	(1,1)
(0,0)	•	•	•	•

4.3 Circular A/G-Reasoning with Σ-Simulation

The structure of circular assume-guarantee reasoning for Σ-simulation is simpler because of the ability to decompose the specification according to Theorem 3.20, from which it follows that $P_1||Q_2 \preceq_\Sigma Q_1$ holds if and only if $P_1||Q_2 \preceq Q_1||Q_2$, analogously $Q_1||P_2 \preceq_\Sigma Q_2$ holds if and only if $Q_1||P_2 \preceq Q_1||Q_2$. We obtain the following structure:

$$\frac{\begin{array}{c} P_1||Q_2 \preceq_\Sigma Q_1 \\ Q_1||P_2 \preceq_\Sigma Q_2 \\ \textit{A/G conditions} \end{array}}{P_1||P_2 \preceq_\Sigma Q_1||Q_2}. \tag{4.17}$$

In contrast to the A/G-rule of Theorem 4.2, Σ-simulation allows us to compare automata of arbitrary alphabets. The definition of Σ-simulation is split into three cases as it may be that a label is exclusively to the alphabet of system, exclusive to the specification, or common to both. This is also reflected in adaption of the A/G conditions (4.11) to Σ-simulation. The A/G conditions must now ensure not only the nonblocking of labels common to both specifications and subsystems, but also the nonblocking of labels that are not part of the system. We get the following theorem:

Theorem 4.5 (A/G-Σ-simulation)**.** *Consider LTSs P_1, P_2, Q_1 and Q_2 with*

$$P_1||Q_2 \quad \preceq_\Sigma \quad Q_1 \textit{ and} \tag{4.18}$$

$$Q_1||P_2 \quad \preceq_\Sigma \quad Q_2. \tag{4.19}$$

If there exist simulation relations R_1 for (4.18) and R_2 for (4.19) such that for all p_1, p_2, q_1, q_2 with $((p_1,q_2),q_1) \in R_1$, $((q_1,p_2),q_2) \in R_2$ and $\alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2}$ there $\exists q_1' : q_1 \xrightarrow{\alpha} q_1'$ or $\exists q_2' : q_2 \xrightarrow{\alpha} q_2'$ whenever:

(i) $\alpha \in \Sigma_{P_1} \backslash \Sigma_{P_2}$ *and* $p_1 \xrightarrow{\alpha} p_1'$,

(ii) $\alpha \in \Sigma_{P_2} \backslash \Sigma_{P_1}$ *and* $p_2 \xrightarrow{\alpha} p_2'$, *or*

(iii) $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$ *and* $p_1 \xrightarrow{\alpha} p_1'$ *and* $p_2 \xrightarrow{\alpha} p_2'$, *or*

(iv) $\alpha \notin \Sigma_{P_1} \cup \Sigma_{P_2}$,

then there is a simulation relation for $P_1 || P_2 \preceq_\Sigma Q_1 || Q_2$ given by

$$R = \{((p_1,p_2),(q_1,q_2)) | ((p_1,q_2),q_1) \in R_1 \wedge ((q_1,p_2),q_2) \in R_2\}. \tag{4.20}$$

Proof. In the following discussion we will identify what the A/G-conditions have to guarantee in order for the proof rule to be sound.

(i) $\alpha \in (\Sigma_{P_1} \cup \Sigma_{P_2}) \cap (\Sigma_{Q_1} \cup \Sigma_{Q_2})$ and $(p_1,p_2) \xrightarrow{\alpha} (p_1',p_2')$:

(a) If $\alpha \in \Sigma_{P_1} \backslash \Sigma_{P_2}$ then $(p_1, p_2) \xrightarrow{\alpha} (p_1', p_2')$ if and only if $p_1 \xrightarrow{\alpha} p_1'$ and $p_2' = p_2$.

(α) If $\alpha \in \Sigma_{Q_1} \backslash \Sigma_{Q_2}$ then there exists a transition $(p_1, q_2) \xrightarrow{\alpha} (p_1', q_2')$ with $q_2' = q_2$ and $((p_1, q_2), q_1) \in R_1$ implies with Def. 3.1(i) that q_1' exists with $q_1 \xrightarrow{\alpha} q_1'$ and $((p_1', q_2'), q_1') \in R_1$. Since $\alpha \notin \Sigma_{Q_2}$ and $((q_1, p_2), q_2) \in R_2$ it follows from Def. 3.1(iii) that there is a transition $(q_1, p_2) \xrightarrow{\alpha} (q_1', p_2)$ with $((q_1', p_2), q_2) \in R_2$. By definition of the parallel composition there is a transition $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ with $q_2' = q_2$, and therefore $((p_1', p_2'), (q_1', q_2')) \in R$.

(β) If $\alpha \in \Sigma_{Q_2} \backslash \Sigma_{Q_1}$, a symmetric argument to (α) shows that $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ with $q_1' = q_1$ and $((p_1', p_2'), (q_1', q_2')) \in R$.

(γ) Otherwise $\alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2}$. From $((p_1, q_2), q_1) \in R_1$ follows from Def. 3.1(i) that if $q_2 \xrightarrow{\alpha} q_2'$ then also $q_1 \xrightarrow{\alpha} q_1'$ and $((p_1', q_2'), q_1') \in R_1$. If so, then $((q_1, p_2), q_2) \in R_2$ imply with Def. 3.1(i) that $((q_1', p_2'), q_2') \in R_2$, and therefore $((p_1', p_2'), (q_1', q_2')) \in R$.

In the case that $q_1 \xrightarrow{\alpha} q_1'$, $((q_1, p_2), q_2) \in R_2$ imply with Def. 3.1(i) that $q_2 \xrightarrow{\alpha} q_2'$ with $((q_1', p_2'), q_2') \in R_2$. Then according to the above, $((p_1', p_2'), (q_1', q_2')) \in R$.

This is a case where the A/G-conditions are needed to ensure that either q_1' or q_2' exist. Here, $\alpha \in (\Sigma_{P_1} \backslash \Sigma_{P_2}) \cap (\Sigma_{Q_1} \cap \Sigma_{Q_2})$ and $p_1 \xrightarrow{\alpha} p_1'$ and the A/G-conditions must imply that either $q_1 \xrightarrow{\alpha} q_1'$ or $q_2 \xrightarrow{\alpha} q_2'$.

(b) If $\alpha \in \Sigma_{P_2} \backslash \Sigma_{P_1}$, a symmetric argument to (b) proves that $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ with $((p_1', p_2'), (q_1', q_2')) \in R$.

For (γ), this is a case where the A/G-conditions are needed to ensure that either q_1' or q_2' exist. Here, $\alpha \in (\Sigma_{P_2} \backslash \Sigma_{P_1}) \cap (\Sigma_{Q_1} \cap \Sigma_{Q_2})$ and $p_2 \xrightarrow{\alpha} p_2'$ and the A/G-conditions must imply that either $q_1 \xrightarrow{\alpha} q_1'$ or $q_2 \xrightarrow{\alpha} q_2'$.

(c) Otherwise $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$ and there are both $p_1 \xrightarrow{\alpha} p_1'$ and $p_2 \xrightarrow{\alpha} p_2'$.

(α) If $\alpha \in \Sigma_{Q_1} \backslash \Sigma_{Q_2}$ then this implies a transition $(p_1, q_2) \xrightarrow{\alpha} (p_1', q_2')$ with $q_2' = q_2$ and $((p_1, q_2), q_1) \in R_1$ implies with Def. 3.1(i) that q_1' exists with $q_1 \xrightarrow{\alpha} q_1'$ and $((p_1', q_2'), q_1') \in R_1$. Since $\alpha \notin \Sigma_{Q_2}$ and $((q_1, p_2), q_2) \in R_2$ it follows from Def. 3.1(iii) that there is a transition $(q_1, p_2) \xrightarrow{\alpha} (q_1', p_2')$ with $((q_1', p_2'), q_2) \in R_2$. By definition of parallel composition, $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ with $q_2' = q_2$, and therefore $((p_1', p_2'), (q_1', q_2')) \in R$.

(β) If $\alpha \in \Sigma_{Q_2} \backslash \Sigma_{Q_1}$, a symmetric argument to (α) shows that $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ with $q_1' = q_1$ and $((p_1', p_2'), (q_1', q_2')) \in R$.

(γ) Otherwise $\alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2}$. From $((p_1, q_2), q_1) \in R_1$ follows from Def. 3.1(i) that if $q_2 \xrightarrow{\alpha} q_2'$ then also $q_1 \xrightarrow{\alpha} q_1'$ and $((p_1', q_2'), q_1') \in R_1$. If so, then $((q_1, p_2), q_2) \in R_2$ implies with Def. 3.1(i) that $((q_1', p_2'), q_2') \in R_2$, and therefore $((p_1', p_2'), (q_1', q_2')) \in R$.

In the case that $q_1 \xrightarrow{\alpha} q_1'$, $((q_1,p_2),q_2) \in R_2$ implies with Def. 3.1(i) that $q_2 \xrightarrow{\alpha} q_2'$ with $((q_1',p_2'),q_2') \in R_2$. Then according to the above, $((p_1',p_2'),(q_1',q_2')) \in R$.
This is a case where the A/G-conditions are needed to ensure that either q_1' or q_2' exist. Here, $\alpha \in (\Sigma_{P_1} \cap \Sigma_{P_2}) \cap (\Sigma_{Q_1} \cap \Sigma_{Q_2})$ and $p_1 \xrightarrow{\alpha} p_1'$ and $p_2 \xrightarrow{\alpha} p_2'$ and the A/G-conditions must imply that either $q_1 \xrightarrow{\alpha} q_1'$ or $q_2 \xrightarrow{\alpha} q_2'$.

(ii) $\alpha \in (\Sigma_{Q_1} \cup \Sigma_{Q_2}) \backslash (\Sigma_{P_1} \cup \Sigma_{P_2})$:

(a) If $\alpha \in \Sigma_{Q_1} \backslash \Sigma_{Q_2}$ then $((p_1,q_2),q_1) \in R_1$ implies with Def. 3.1(ii) that q_1' exists with $q_1 \xrightarrow{\alpha} q_1'$ and $((p_1,q_2),q_1') \in R_1$. Since $\alpha \notin \Sigma_{Q_2}$ and $((q_1,p_2),q_2) \in R_2$ it follows from Def. 3.1(iii) that $(q_1,p_2) \xrightarrow{\alpha} (q_1',p_2)$ with $((q_1',p_2),q_2) \in R_2$. By definition of the parallel composition, $(q_1,q_2) \xrightarrow{\alpha} (q_1',q_2')$ with $q_2' = q_2$, and therefore $((p_1,p_2),(q_1',q_2')) \in R$.

(b) If $\alpha \in \Sigma_{Q_2} \backslash \Sigma_{Q_1}$, a symmetric argument to (a) shows that $(q_1,q_2) \xrightarrow{\alpha} (q_1',q_2')$ with $q_1' = q_1$ and $((p_1,p_2),(q_1',q_2')) \in R$.

(c) Otherwise $\alpha \in \Sigma_{Q_2} \cap \Sigma_{Q_1}$, and since it holds that $((p_1,q_2),q_1) \in R_1$ and $((q_1,p_2),q_2) \in R_2$, either both q_1' and q_2' exist, from which it follows that $((p_1,p_2),(q_1',q_2')) \in R$, or neither exist. *This is a case where additional conditions are needed to ensure that either q_1' or q_2' exist. Here, $\alpha \in (\Sigma_{Q_1} \cap \Sigma_{Q_2}) \backslash (\Sigma_{P_1} \cup \Sigma_{P_2})$ and C must imply that either $q_1 \xrightarrow{\alpha} q_1'$ or $q_2 \xrightarrow{\alpha} q_2'$.*

(iii) $\alpha \in (\Sigma_{P_1} \cup \Sigma_{P_2}) \backslash (\Sigma_{Q_1} \cup \Sigma_{Q_2})$ and $(p_1,p_2) \xrightarrow{\alpha} (p_1',p_2')$:

(a) If $\alpha \in \Sigma_{P_1} \backslash \Sigma_{P_2}$ then $(p_1,p_2) \xrightarrow{\alpha} (p_1',p_2')$ if and only if $p_1 \xrightarrow{\alpha} p_1'$ and $p_2' = p_2$. Then $((p_1,q_2),q_1) \in R_1$ implies with Def. 3.1(iii) that $((p_1',q_2),q_1) \in R_1$. Since $p_2' = p_2$, $((q_1,p_2'),q_2) \in R_2$ and therefore $((p_1',p_2'),(q_1,q_2)) \in R$.

(b) If $\alpha \in \Sigma_{P_2} \backslash \Sigma_{P_1}$ then in symmetry to (a): $((p_1',p_2'),(q_1,q_2)) \in R$.

(c) Otherwise $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$. Then $p_1 \xrightarrow{\alpha} p_1'$ and $p_2 \xrightarrow{\alpha} p_2'$. $((p_1,q_2),q_1) \in R_1$ implies that $((p_1',q_2),q_1) \in R_1$ holds, and $((q_1,p_2),q_2) \in R_2$ implies $((q_1,p_2'),q_2) \in R_2$. Therefore $((p_1',p_2'),(q_1,q_2)) \in R$.

In the above discussion, four cases were isolated where the A/G-conditions must ensure that either $q_1 \xrightarrow{\alpha} q_1'$ or $q_2 \xrightarrow{\alpha} q_2'$. These are precisely the A/G-conditions stated in the hypothesis, which concludes the proof. □

Note that R doesn't necessarily contain the initial states. As with A/G-simulation, it follows from the definitions of simulation and parallel composition that the conditions (i)–(iv) are fulfilled for any simulation relation that witnesses $P_1||P_2 \preceq Q_1||Q_2$. The checking for containment of the initial states is done as for simulation in Sect. 4.2.3,

procedure *CheckAGSigmaSimulationSeparate*
Input: labeled transition systems P_1, P_2, Q_1, Q_2,
Output: A/G-simulation relations R_1, R_2

$R_1 := reach_{P_1||Q_2||Q_1}$, $R_2 := reach_{Q_1||P_2||Q_2}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_2}(R_2)$
$R_1 := R_1 \backslash \{(p_1, q_2, q_1) | \exists \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} :$
$\nexists q_1' : (q_1 \xrightarrow{\alpha} q_1') \wedge \nexists q_2' : (q_2 \xrightarrow{\alpha} q_2') \wedge [(\alpha \in \Sigma_{P_1} \wedge \exists p_1' : p_1 \xrightarrow{\alpha} p_1') \vee \alpha \notin (\Sigma_{P_1} \cup \Sigma_{P_2})]\}$
$R_2 := R_2 \backslash \{(q_1, p_2, q_2) | \exists \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} :$
$\nexists q_1' : (q_1 \xrightarrow{\alpha} q_1') \wedge \nexists q_2' : (q_2 \xrightarrow{\alpha} q_2') \wedge [(\alpha \in \Sigma_{P_2} \wedge \exists p_2' : p_2 \xrightarrow{\alpha} p_2') \vee \alpha \notin (\Sigma_{P_1} \cup \Sigma_{P_2})]\}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_2}(R_2)$

Figure 4.7: Algorithm for checking assume-guarantee Σ-simulation by separate trimming

with $R_1' = R_1$ and $R_2' = R_2$. An algorithm for checking A/G-simulation by separate trimming is shown in Fig. 4.7. It is easy to see that R_1 and R_2 fulfill conditions (i)–(iv) after the trimming. Figure 4.8 show the algorithm for composite trimming adapted to Σ-simulation.

procedure *CheckAGSigmaSimulationComposite*
Input: labeled transition systems P_1, P_2, Q_1, Q_2
Output: A/G-simulation relations R_1, R_2

$R_1 := reach_{P_1||Q_2||Q_1}$, $R_2 := reach_{Q_1||P_2||Q_2}$
$R_1 := R_1 \backslash \{(p_1, q_2, q_1) | \exists \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} :$
$\nexists q_1' : (q_1 \xrightarrow{\alpha} q_1') \wedge \nexists q_2' : (q_2 \xrightarrow{\alpha} q_2') \wedge [(\alpha \in \Sigma_{P_1} \setminus \Sigma_{P_2} \wedge \exists p_1' : p_1 \xrightarrow{\alpha} p_1') \vee \alpha \notin (\Sigma_{P_1} \cup \Sigma_{P_2})]\}$
$R_2 := R_2 \backslash \{(q_1, p_2, q_2) | \exists \alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2} :$
$\nexists q_1' : (q_1 \xrightarrow{\alpha} q_1') \wedge \nexists q_2' : (q_2 \xrightarrow{\alpha} q_2') \wedge [(\alpha \in \Sigma_{P_2} \setminus \Sigma_{P_2} \wedge \exists p_2' : p_2 \xrightarrow{\alpha} p_2') \vee \alpha \notin (\Sigma_{P_1} \cup \Sigma_{P_2})]\}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_2}(R_2)$
$D_{P_1} = \{(q_1, q_2, \alpha) | \alpha \in \Sigma_{P_1} \cap \Sigma_{P_2} \cap \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_1 : (p_1, q_2, q_1) \in R_1 \wedge$
$p_1 \xrightarrow{\alpha} p_1' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}$
$D_{P_2} = \{(q_1, q_2, \alpha) | \alpha \in \Sigma_{P_1} \cap \Sigma_{P_2} \cap \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_2 : (q_1, p_2, q_2) \in R_2 \wedge$
$p_2 \xrightarrow{\alpha} p_2' \wedge \nexists q_1' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}.$
if $D_{P_1} \cap D_{P_2} \neq \emptyset$
$R_1 := R_1 \backslash \{(p_1, q_2, q_1) | \exists \alpha, p_1' : p_1 \xrightarrow{\alpha} p_1' \wedge (q_1, q_2, \alpha) \in D_{P_2}\}$
$R_2 := R_2 \backslash \{(q_1, p_2, q_2) | \exists \alpha, p_2' : p_2 \xrightarrow{\alpha} p_2' \wedge (q_1, q_2, \alpha) \in D_{P_1}\}$
$R_1 := GetSimRel_{P_1||Q_2,Q_1}(R_1)$
$R_2 := GetSimRel_{Q_1||P_2,Q_2}(R_2)$
fi

Figure 4.8: Algorithm for checking assume-guarantee Σ-simulation by composite trimming of R_1 and R_2

Example 4.7. *Consider a reactor with two inlets and one outlet as shown in Fig. 4.9. The task of a controller is to transfer raw material from Tank 1 to Tank 3, represented by a label T13, then from Tank 2 to Tank 3, represented by T23, and finally drain the mixture to Tank 4, symbolized by label T34. A discrete model for Tank 3 is shown in Fig. 4.10(a). It contains error states for unmodelled behavior, i.e. states with a self-loop with label error. The error states are reachable when an empty tank is drained or a full tank filled. The controller model is shown in Fig. 4.10(b).*

The verification task is to show that no error states are reachable, and that the mixing sequence is preserved, i.e. that first tank 1 is drained and then tank 2. The automaton for this specification is shown in Fig. 4.11. The label error is in the alphabet $\Sigma_Q = \{T13, T23, error\}$ *of Q, but since there are no transitions with that label, it is always forbidden. For the assume-guarantee proof, the specification Q was manually decomposed into specifications* Q_1 *and* Q_2 *for the tank and the controller such that* $Q_1||Q_2 \preceq Q$*. Figures 4.12(a) and 4.12(b) show the automata.*

Checking the state space of $P_1||Q_2||Q_1$ *and* $Q_1||P_2||Q_2$ *yields that* $D_{P_1} = \{(0,0,T34), (1,1,T34), (2,2,T13), (2,2,T23)\}$ *and* $D_{P_2} = \{\}$*.* R_1 *and* R_2 *are initialized with the states of* $P_1||Q_2||Q_1$ *and* $Q_1||P_2||Q_2$*, and contain no states that could violate the A/G-conditions. The algorithm in Fig. 3.4 is applied to turn them into simulation relations. As a result, both relations contain the initial states and the specifications* Q_1 *and* Q_2 *are fulfilled. Because* $Q_1||Q_2 \preceq Q$ *holds, transitivity implies that the global specification Q also holds.*

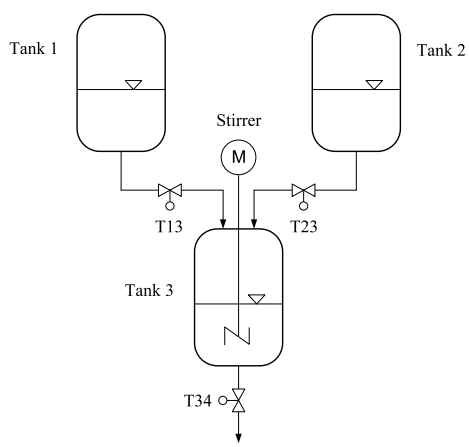
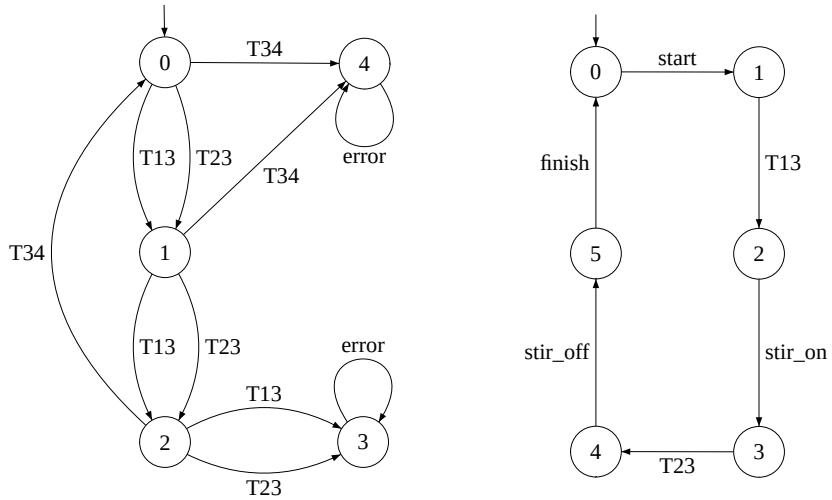


Figure 4.9: Reactor with two raw material tanks



(a) Tank 3 P_1 (b) Controller P_2

Figure 4.10: Labeled transition system models for tank and controller

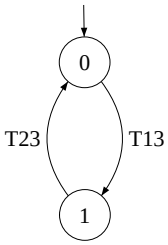
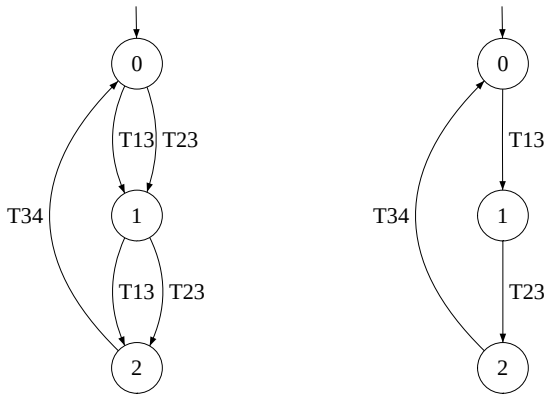


Figure 4.11: Global specification Q with alphabet $\Sigma_Q = \{T13, T23, error\}$



(a) Tank Specification Q_1 (b) Controller Specification Q_2

Figure 4.12: Labeled transition system models for the decomposed specifications

4.4 Related Work

Assume-Guarantee Reasoning Proofs that are based on assertions have been developed independently by Jones [Jon81, Jon83], in the *rely-guarantee* formalism, and Misra and Chandy [MC81] in the *assumption-commitment* framework, see [RBH+01] for a comprehensive discussion. Proofs of this type have been used extensively for discrete and hybrid systems, and the term assume-guarantee reasoning has established itself in the literature concerned with hybrid systems.

An overview of compositional reasoning is given by de Roever in [Roe98], others are available by Berezin et al. in [BCC98] and Xu et al. [XS98]. For a survey of assume-guarantee reasoning for Moore Machines and Reactive Modules see [PT98]. A small comparison of A/G-techniques with SPIN and SMV is given by Pasareanu et al. [PDH99]. They conclude that assumptions are better encoded in the state space than in a formula. A general discussion of the problems of assume-guarantee reasoning and the role of interference is given by Collette and Jones in two recent papers [CJ00, Jon03]. Non-circular A/G-reasoning was applied to agents by Abadi and Lamport [AL93], and improved version of which can be found in [AL95]. *Complete* non-circular and circular A/G-rules for checking LTL properties have been presented by Namjoshi et al. in [NT00], which also contains a small survey. Henzinger et al. have applied A/G-reasoning to reactive systems using MOCHA, a software environment for system specification and verification, for which a tutorial introduction is given in [HQR00]. The framework is based on trace containment [HQR98], and supports hierarchy [AG00]. The approach has been successfully applied to verify an array of 64 dataflow processors [HLQR99]. For A/G-reasoning of liveness properties see the works of McMillan [McM99, McM00]. The A/G-rule presented there was verified by Rushby with the automatic theorem prover PVS [Rus01]. The tool COSPAN provides some functionality to verify refinement of discete, timed systems and a class of hybrid systems [AK96, TAKB96].

A/G-Reasoning based on Simulation Relations In [LS95], simulation-based assertional techniques were compared to process algebraic techniques in the verification of a small but supposedly typical circuit, a Mueller C element. The concept of simulation has been used by McMillan et al. [McM00, JM01], where it is called *refinement*. An assume-guarantee rule for fair simulation of Moore machines has been proposed by Henzinger et al. [HQRT02]. It is stricter than the rule of Theorem 4.2 because it requires non-blocking and finite non-determinism. A more detailed treatment of one of the same authors can be found in [Raj99]. Viswanathan et al. presented assume-guarantee rules based on traces and trace-trees with a semantics that avoid the use of non-propositional reasoning such as induction [VV01]. They show that the rule for simulation by Henzinger et al. [HQRT02] can be derived from their rule for trace-trees.

Part II

Hybrid Systems with Discrete Interaction

Introduction

The integration of digital controllers with a continuously active physical system can result in *discrete-continuous* behavior. It is governed by the ongoing change of the physical component as well as instantaneous changes triggered by the digital logic at discrete events. The former phenomenon can be efficiently modelled with differential equations, and the latter with state-transition graphs. A system that exhibits both types of behaviors is called a *hybrid system*, and their fusion into a single formalism has resulted in the paradigm of hybrid automata, as introduced in 1993 by Alur et al. [ACHH93].

Hybrid systems are notoriously complex to analyze, and fundamental properties, like the reachability of forbidden states, are in general undecidable. For algorithmic verification, one must therefore resort to special classes of hybrid systems, for which these problems are decidable or at least semi-computable, and use these to overapproximate more complex models. Because of this need for computationally manageable systems we are particularly interested in *linear hybrid automata* [Hen96], which have been studied intensely, and for which reachability and simulation are semi-computable with linear predicates. While the following chapters refer to hybrid automata in general, their application in practice involves linear hybrid automata, which which are also used in PHAVer, a verification tool that are presented in Chapter 11.

It is well known, and appreciated by anyone who has tried to verify hybrid systems in practice, that the computational cost of verification increases exponentially with the number of interacting components of the system, and with the number of continuous variables. Two widely recognized and fundamental methods to counter this phenomenon, which is referred to as the *state-explosion problem*, are *abstraction* and *compositional reasoning*. We use simulation relations to apply both methods, and discuss the limitations of traditional semantics in this respect.

Following an approach by Henzinger in [Hen96], we define simulation for hybrid automata based on their timed transition system (TTS) semantics. While some continuous variables might only be relevant for the timing of the system, others are associated with a particular meaning, e.g., the level of a liquid filling a tank. This implies that if the state q of a specification simulates the state p of a system, the values of those variables should also be considered equivalent in p and q, e.g., the value of the level variable in the tank model should be identical to the value of any variable referring to the tank level in the specification. This is achieved by requiring that the

states in the simulation relation are in a given equivalence relation $\approx$. This simulation with equivalence is called $\approx$-simulation. TTS-semantics allow us to apply many of the definitions and algorithms from Part I, with special attention to compositionality. To apply compositional reasoning at the level of hybrid automata it is necessary that the TTS-semantics and parallel composition are commutative. We show that this is the case if there are no shared variables. We give symbolic formulations of an algorithm for a computing simulation relation between hybrid systems. It employs operations in $\mathbb{R}^n$, and we discuss its shortcomings and discuss ways to improve the performance. While the algorithm is similar to the one in Sect. 3.3, its formulation in $\mathbb{R}^n$ illustrates the complexity that results from by the difference operations involved.

In the following chapter, we formally define hybrid automata and their labeled transition system semantics. In Chap. 6 we define $\approx$-simulation and show that TTS-semantics and parallel composition are commutative in the absence of shared variables. A symbolic formulation of an algorithm to compute the simulation relation is given, and we discuss approaches to speed up the computation. Compositional proof rules are presented in their symbolic form in Chap. 7, and illustrated by an example. We present our conclusions for Part II at the end of this thesis in Sect. 12.2, pp. 168.

Chapter 5

Modeling with Hybrid Automata

Hybrid automata are a paradigm for models of continuous-discrete behavior. They are based on state-transition systems with a special structure that reflects the continuous and discrete components of the behavior. The system is considered to have a finite set of modes, called *locations*, in which the continuous dynamics of the variables is given by functions over time, usually defined implicitly by differential equations. An instantaneous change in the variables or dynamics is modeled with a *transition* to another location. A variety of hybrid automata concepts have been proposed since their initial publication in 1993 [ACHH93]. We present our own blend of these concepts – not just to add another species to the zoo, but in an effort to obtain a formalism that most allows us to apply the methods for labeled transition systems from the previous part as directly as possible. However, the modifications are of a formal rather than conceptual nature. Our choices are motivated in a detailed comparison in Sect. 5.4.

In the following section we present our hybrid automaton model and recall the definition of linear hybrid automata in Sect. 5.2. In Sect. 5.3 we recall the labeled transition system semantics from [Hen96], also called *timed transition system semantics*, that will allow us to apply the methods for labeled transition systems to hybrid systems. The chapter finishes with a summary of related work in Sect. 5.4.

5.1 Hybrid Automata

Our formal definition of hybrid automata blends the classic definitions in [ACHH93, ACH+95, Hen96]. It differs from those since we attribute a label to each discrete transition without requiring stutter transitions, see Sect. 5.4 for more detail on related work. We recall basic notions of variable valuations and activities before proceeding with the definition of hybrid automata.

A variable is an identifier that is associated with a real number, its ***value***. This is formally captured by a mapping, called a ***valuation***, just like a n-dimensional vector $x^T = (x_1, \ldots, x_n)$ can be interpreted as a mapping from an index $1, \ldots, n$ to a value. The continuous change of a variable over time is defined by an ***activity***. When no confusion

arises we will sometimes simplify the notation by writing the variable instead of its valuation, e.g., x instead of $v(x)$.

Definition 5.1 (Valuation, Activity, Projection)**.** [ACHH93] *Given a set Var of variables, a* **valuation** *is a function* $v : Var \to \mathbb{R}$. *Let* $V(Var)$ *denote the set of valuations over Var, where* $V(\emptyset) = \{\emptyset \to \emptyset\}$. *An* **activity** *is a function* $f : \mathbb{R}^{\geq 0} \to V(Var)$. *Let* $Ats(Var)$ *denote the set of activities over the variables in Var. Given a set of variables* $Var' \subseteq Var$, *let the* **projection** $v' = v{\downarrow}_{Var'}$ *be the valuation over* Var' *defined by* $v'(x) = v(x)$ *for all* $x \in Var'$. *The extension to activities is straightforward.*

Remark: Usually the set of activities in a location of the system is considered to be infinitely differentiable [ACHH93], or time-invariant [ACH+95, NOSY92, Hen96].[1] Neither is required for the results in this thesis.

Hybrid automata are state-transition systems with variables that can change continuously over time, or at discrete events. A state is a pair of a location and a valuation over the variables. The evolution of the variables is constrained to an invariant set for each location. A discrete transition from one state to another can take place if and only if source state and target state are in a continuous transition relation, which is associated with every transition. A behavior only belongs to the automaton if it originates in a set of initial states.

Definition 5.2 (Hybrid Automaton)**.** [ACH+95, Hen96] *A* **hybrid automaton** $H = (Loc, Var, Lab, \to, Act, Inv, Init)$ *has the following components:*

- *A finite set Loc of* **locations**.
- *A finite set Var of variables. A pair* (l, v) *of a location and a valuation of the variables is a* **state** *of the automaton. The set of all states* $S_H = Loc \times V(Var)$ *is called the* **state space**.
- *A finite set Lab of synchronization labels,*
- *A finite set of* **transitions** $\to \subseteq Loc \times Lab \times 2^{V(Var) \times V(Var)} \times Loc$. *A transition* $(l, a, \mu, l') \in \to$ *is also written as* $l \xrightarrow{a,\mu}_H l'$. l *and* l' *are called* **source** *and* **target** *locations, and* μ *is called the* **continuous transition relation**.
- *A mapping* $Act : Loc \to 2^{Ats(Var)}$ *from locations to sets of activities.*
- *A mapping* $Inv : Loc \to 2^{V(Var)}$ *from locations to sets of valuations, called* **invariants**.
- *A set* $Init \subseteq Loc \times V(Var)$ *of initial states that lie inside the invariants, i.e.,* $(l, v) \in Init \Rightarrow v \in Inv(l)$.[2]

[1] A set S of activities is ***time-invariant*** if for all $f \in S, d \in \mathbb{R}^{\geq 0} : f'(t) = f(t+d) \in S$.

[2] We admit an empty set of initial states because it can occur as a result of parallel composition, when the sets of initial states or the invariants of two automata are disjoint.

Remark: Some automaton models, like the timed automata in [AD92], define the continuous transition relation with a ***guard*** $G \subseteq V(Var)$ and a set of ***reset*** valuations $R \subseteq V(Var')$ over a set of variables $Var' \subseteq Var$. In that concept, the variables in Var' take any value in R after the transition, and remain unchanged otherwise. This translates into the continuous transition relation

$$\mu = \{(v,v') | v \in G \wedge v{\downarrow}_{Var \setminus Var'} = v'{\downarrow}_{Var \setminus Var'} \wedge v'{\downarrow}_{Var'} \in R\}.$$

We use the following shorthand notation to describe the continuous transition relation: For a binary relation R over a set of variables Var, i.e., $R \subseteq V(Var) \times V(Var)$, we use x' to denote the variable x when it occurs in the second element of a pair in R. E.g., the equation $x + x' = 0$ defines the relation

$$R = \{(v,v') \in V(Var) \times V(Var) | v(x) + v'(x) = 0\}$$

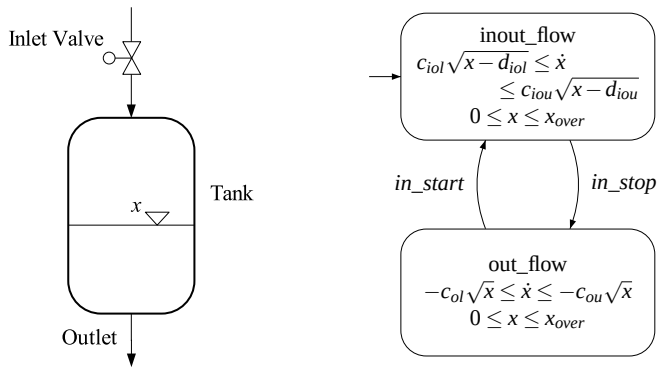
Example 5.1. *Consider a tank with an inlet valve and a constantly open outlet as shown in the schematic in Fig. 5.1(a). A hybrid automaton model of the tank is shown in Fig. 5.1(b). There are two locations, symbolized by rectangular nodes, "inout_flow" for an open inlet valve, and "out_flow" for a closed one. The variable x represents the tank level. The activities in the locations are given implicitly by contraints on the derivative of x. Constants* $c_{iol}, c_{iou} \geq 0$ *determine the lower and upper rate of x, and* $d_{iol}, d_{iou} \geq 0$ *the set of equilibrium points for x if the inlet valve is open. If it is closed, the rate is determined by* $c_{ol}, c_{ou} \geq 0$. *An invariant in both locations constrains x to remain between zero and a constant* x_{over}. *Transitions are represented by directed arcs. A transition with label "in_start" represents a change in dynamics caused by opening the valve, and one with label "in_stop" closing the valve. The variable does not change during the transition, so the continuous transition relation is given by* $x' = x$, *and omitted from the graphical representation. An incoming arc designates the location "inout_flow" as an initial location. I can be labeled with the initial states if they are different from the invariant.*

The semantics of a hybrid automaton is defined by its *runs*, a possibly infinite sequence of discrete transitions and periods of time elpase. When time passes, the continuous variables change according to the activities, but they must remain inside the invariants. Thus, periods of time elapse are labeled with a duration and the activity. A run contains all necessary information about how the automaton changes its states, i.e., labels as well as activities and their durations. Only the runs that start from one of the initial states are considered actual behavior of the automaton, and are called *executions*.

Definition 5.3 (Run, Execution, Reachability)**.** *A* **run** σ *is a finite or infinite sequence of states* (l_i, v_i) *and labels* $\alpha_i \in Lab \cup \left(\mathbb{R}^{\geq 0} \times Ats(Var)\right)$,

$$\sigma = (l_0, v_0) \xrightarrow{\alpha_0} (l_1, v_1) \xrightarrow{\alpha_1} (l_2, v_2) \xrightarrow{\alpha_1} \ldots, \tag{5.1}$$

satisfying that for all $i \geq 0$ *holds* $v_i \in Inv(l_i)$ *and either*



(a) Schematics

(b) Hybrid automaton model with non-linear dynamics

Figure 5.1: Tank T with inlet valve and constant outflow

- $\alpha_i \in Lab$ *and there is a transition* $l_i \xrightarrow{\alpha_i,\mu}_H l_{i+1}$ *with* $(v_i, v_{i+1}) \in \mu$, *or*
- $\alpha_i =: (t_i, f_i)$ *and* $f_i \in Act(l_i)$, $l_i = l_{i+1}$, $f_i(0) = v_i, f_i(t_i) = v_{i+1}$ *and for all* t', $0 \le t' \le t_i$, *holds* $f_i(t') \in Inv(l_i)$.

An **execution** *is a run* σ *that starts in one of the initial states, i.e.,* $(l_0, v_0) \in Init$. *A state* (l, v) *is called* **reachable** *if there exists an execution with* $(l, v) = (l_i, v_i)$ *for some* $i \ge 0$.

Usually a system can be divided into several components, each of which is then modeled by a separate automaton. Their interaction is determined by a *parallel composition* operator that merges them into one single automaton. Hybrid automata interact in two ways: They synchronize on discrete transitions, and they share variables. As discussed previously in Sect. 2.4, shared variables with unrestricted access are not compositional, so that in this Part we limit the compositional analysis to hybrid automata that each have their own set of variables that is disjunct of all others. The compositional analysis of hybrid systems with shared variables requires a more sophisticated model, and will be the subject of Part III.

In the following, hybrid automata are considered to interact by synchronizing on common labels. If an automaton does not participate in a transition, its variables remain constant, which leads to the following composition operator:

Definition 5.4 (Parallel Composition)**.** *Given hybrid automata* $H_i = (Loc_i, Var_i, Lab_i, \rightarrow_i, Act_i, Inv_i, Init_i), i = 1, 2$, *their* **parallel composition** $H_1 || H_2$ *is the hybrid automaton* $H = (Loc, Var, Lab, \rightarrow_H, Act, Inv, Init)$ *with*

- $Loc = Loc_1 \times Loc_2$, $Var = Var_1 \cup Var_2$, $Lab = Lab_1 \cup Lab_2$,
- $f \in Act(l_1, l_2)$ *iff* $f\downarrow_{Var_i} \in Act_i(l_i)$ *for* $i = 1, 2$,

- $v \in Inv(l_1, l_2)$ *iff* $v{\downarrow}_{Var_i} \in Inv_i(l_i)$ *for* $i = 1, 2$, *and*
- $(l_1, l_2) \xrightarrow{a,\mu}_H (l_1', l_2')$ *with* $\mu = \{(v, v') | (v{\downarrow}_{Var_i}, v'{\downarrow}_{Var_i}) \in \mu_i, i = 1, 2\}$ *iff for* $i = 1, 2$*:*
  - $a \in Lab_i$ *and* $l_i \xrightarrow{a,\mu_i}_i l_i'$, *or*
  - $a \notin Lab_i$ *and* $l_i = l_i'$, $\mu_i = \{(v, v') | v{\downarrow}_{Var_i} = v'{\downarrow}_{Var_i}\}$.
- $((l_1, l_2), v) \in Init$ *iff* $(l_i, v{\downarrow}_{Var_i}) \in Init_i$ *for* $i = 1, 2$ *and* $v \in Inv(l_1, l_2)$.

The following example shall illustrate how a system is naturally modeled by several interacting components, each corresponding to a hybrid automaton, and the system as whole corresponding to their parallel composition.

Example 5.2. *Consider the tank with an inlet valve T from Fig. 5.1(b), equipped with two discrete level sensors L_h and L_l, positioned at levels x_h and x_l. We model the tank with sensors in a modular fashion with separate models for the sensors, and combine them by parallel composition. Figures 5.2(a) and 5.2(b) show hybrid automaton models of the sensors. They only have a single location "idle", in which the variable x is unconstrained, i.e., $Act(idle) = Ats(x)$ and $Inv(idle) = \mathbb{R}$. If the level is above x_h, L_h has a self-looping transition with label x_high, otherwise one with label x_nhigh. L_l works correspondingly. The composed system $T_s = T||L_l||L_h$ is shown in Fig. 5.2(c). The sets of valuations of x in invariants, activities and initial states were intersected, but since x is unconstrained in L_h and L_l, they are the same as in the original model T. The labels are not in the alphabet of T, so the transitions are added with the constraint $x' = x$ (not shown). Clearly, the modular model is much more convenient and intelligible than the composition.*

In order to model shared variables into the framework of discrete interactions, access of other automata to the variable can be incorporated using synchronization labels. This implies that for a hybrid automaton with a finite number of discrete transitions there can be only a finite number of values for each shared variable. As a result, such a set of interacting hybrid automata is unable to perform calculations over an infinite number of values, as it is the case, e.g., for continuous feedback controllers. Still, it can be useful to model a system with hybrid automata with shared variables. The fact that this is not compositional only means that the automata that share a variable should not be decomposed in a compositional proof.

5.2 Linear Hybrid Automata

We are particularly interested in hybrid automata that can be analyzed in an efficient manner. Since hybrid automata are systems with an infinite state space, this naturally involves symbolic computations on sets of states. Our focus lies on polyhedra as a symbolic representation, since very efficient algorithms are available, like the Motzkin

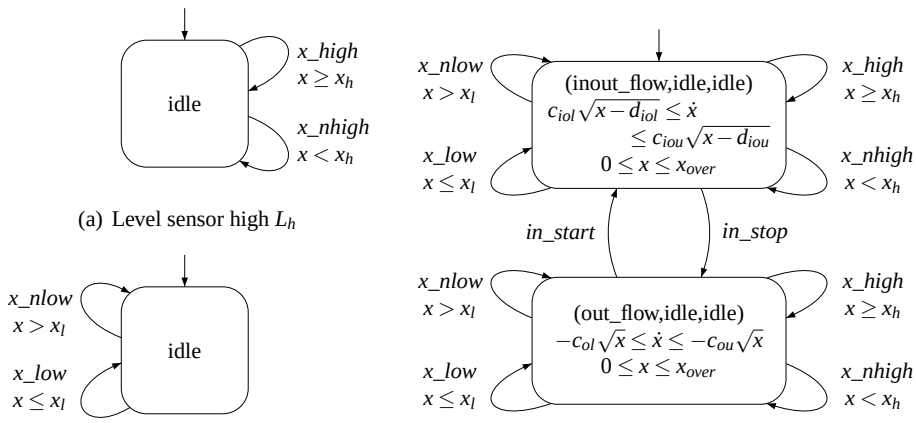


Figure 5.2: Tank model with level sensors

double description method [MRTT53] and its enhancements. For an overview on algorithms refer to [Fuk04] and implementations, as well as further references, can be found in [Wil97, BRZ04]. Our verification tool is based on polyhedral computations, and will be described in Part 11. A ***polyhedron*** is a subset of $\mathbb{R}^n$ that is the set of solutions to a finite system of linear inequalities, called *linear constraints* [Fuk04] :

Definition 5.5 (Linear Constraint, Linear Predicate)**.** *A* **linear expression** *over a set of variables Var is of the form* $\sum_i a_i x_i + b$*, with* $a_i, b \in \mathbb{Z}$*,* $x_i \in Var$*. A* **linear constraint** *over Var is of the form* $e \bowtie 0$*, where e is a linear expression and the sign* $\bowtie$ *is in* $\{<, \leq\}$*.For a given valuation v over Var, a linear constraint* ϕ *defines a boolean value* $\phi(v)$ *that yields whether* ϕ *is satisfied or not, i.e., whether* $\sum_i a_i v(x_i) + b \bowtie 0$ *holds. A* **convex linear predicate** *is a finite conjunction of linear constraints. A* **non-convex linear predicate***, or* **linear predicate***, is a finite disjunction of convex linear predicates.*

A very important class of hybrid automata are *linear hybrid automata* (LHA).[3] For LHAs, reachability and simulation are computable with exactness using linear predicates. As we will show in Part III, our definition of simulation is also compositional for LHAs with convex invariants. They are defined as follows:

Definition 5.6 (Linear Hybrid Automaton)**.** [Hen96][4] *A* **linear hybrid automaton** *(LHA) is a hybrid automaton in which the invariants and the continuous transition*

[3]The term linear hybrid automaton is ambiguous since it is also used, e.g., in [LPY01], to describe hybrid automata whose dynamics are given by a linear time-invariant differential equation system, i.e., $\dot{x} = Ax$ or $\dot{x} = Ax + Bu$. We call such dynamics ***affine***.

[4]Older definitions of linear hybrid automata in [ACHH93, ACH+95] only admit dynamics with a constant derivative given for each location.

relation are given by linear predicates over its variables, and the activities are given by linear predicates over the time derivatives of the variables.

Linear hybrid automata are readily used as overapproximations of hybrid automata with complex dynamics, see Sect. 5.4 for related work. This approximation is asymptotically complete, i.e., any hybrid automaton can be approximated arbitrarily close with a LHA [HHWT98]. The following example shall illustrate such an overapproximation:

Example 5.3. *Consider the nonlinear tank model from Ex. 5.1. Figure 5.3 shows a linear hybrid automaton $\hat{T}$ that overapproximates the non-linear dynamics in an interval $x \in [0, x_M]$ by upper and lower bounds of n intervals. The choice of bounds was made under the assumption that $f_{iol}(x) \leq f_{iou}(x)$, $f_{ol}(x) \leq f_{ol}(x)$, and that they are monotonously increasing, respectively decreasing, with x. The characteristic curve of Ex. 5.1 is given by:*

$$\begin{aligned} f_{iol}(x) &= c_{iol}\sqrt{x - d_{iol}}, \\ f_{iou}(x) &= c_{iou}\sqrt{x - d_{iou}}, \\ f_{ol}(x) &= c_{ol}\sqrt{x}, \\ f_{ou}(x) &= c_{ou}\sqrt{x}, \end{aligned}$$

where $c_{iol}, c_{iou}, c_{ol}, c_{ou}, d_{iol}, d_{iou}$ are constants to be identified by the physical setup. The automaton shown assumes initial states within the interval $[x_M/n, 2x_M/n]$, and the appropriate locations are designated with incoming arcs as having initial states.

It is worthwhile to note that discrete-time piecewise affine systems, or more precisely hybrid automata with linear invariants, linear continuous transition relations and discrete-time linear dynamics, are also linear hybrid automata. This underlines the relevance of the class in applications related to control systems. A discrete-time piecewise affine system can be embedded in a linear hybrid automaton model as follows:

Example 5.4. *A linear hybrid automaton model of a discrete-time, linear time-invariant differential equation system is shown in Fig. 5.4. A clock t is introduced to measure the sampling time δ. When t has reached δ, the invariant forces the transition, the variables x are updated according to the dynamics and the clock is reset to zero. If the hybrid automaton is only composed with other sampled systems, the clock can be omitted as they will synchronize on the label "tick".*

Linear hybrid automata have received considerable attention, and a model-checking tool called HYTECH is available [HHWT97]. A symbolic model-checking procedure for a temporal logic with stopwatches was introduced in [AHH96].

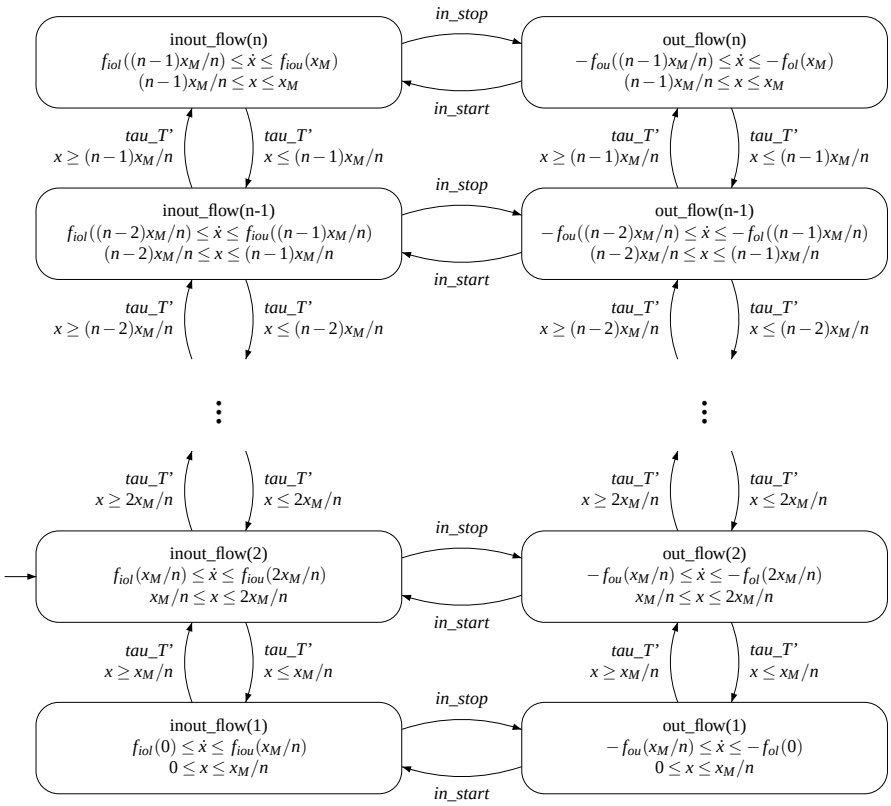


Figure 5.3: LHA-approximation $\hat{T}$ of tank T from Fig. 5.1(b)

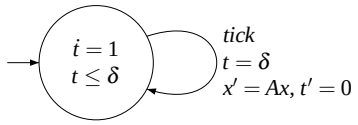


Figure 5.4: Linear hybrid automaton with linear discrete time dynamics

5.3 Timed Transition System Semantics

Temporal logics like CTL* [CE81] are widely used to define and reason about properties of automata, and are based on runs. These properties can be divided into ***safety*** and ***liveness*** properties. Intuitively, a safety property describes that something bad can never happen, while a liveness property tells whether something good will always occur. Rather than with runs, the safe semantics of a hybrid automaton can be defined with an infinite labeled transition system [Hen96]. This considerably simplifies many proofs, allows us to carry over the results of Part I, and so provides the basis for our further work.

Definition 5.7 (Timed Transition System)**.** [Hen96] *The* **timed transition system** *of a hybrid automaton* $H = (Loc, Var, Lab, \rightarrow_H, Act, Inv, Init)$ *is the labeled transition system* $[\![H]\!] = (S_H, Lab \cup \mathbb{R}^{\geq 0}, \rightarrow_{[\![H]\!]}, Init)$ *where*

- $(l,v) \xrightarrow{a}_{[\![H]\!]} (l',v')$ *if and only if* $l \xrightarrow{a,\mu}_H l', (v,v') \in \mu, v \in Inv(l), v' \in Inv(l')$ *(discrete transitions),*

- $(l,v) \xrightarrow{t}_{[\![H]\!]} (l,v')$ *if and only if there exists an activity* $f \in Act(l)$*, with* $f(0) = v, f(t) = v'$ *such that for all* $t', 0 \leq t' \leq t$ *holds* $f(t') \in Inv(l)$ *(timed transitions).*

We refer to the subset of $\rightarrow_{[\![H]\!]}$ with labels in Lab as the ***discrete transition relation***, and the one with labels in $\mathbb{R}^{\geq 0}$ as the ***timed transition relation*** $\overset{t}{\rightsquigarrow}$. The set of valuations (v,v',t) with $((l,v),(l,v')) \in \overset{t}{\rightsquigarrow}$ is also denoted as $\rightsquigarrow (l)$.

Example 5.5. *Figure 5.5 shows the timed transition relation for a location q with an invariant* $u_{min} \leq u \leq u_{max}$ *and a set of activities given by* $a \leq \dot{u} \leq b$, $0 < a < b$*. Note how by definition the source and target states are the same for* $t = 0$*. The front face of the tetrahedron is the set of successors* $u'(t)$ *of the initial state* $u = u_{min}$*.*

A run of the timed transition system is abstraction of a run of the hybrid automaton because of the existential quantification over timed transitions. Formally this can be expressed as follows:

Proposition 5.8. *There is a run* $\sigma = p_0 \xrightarrow{\alpha_0} p_1 \xrightarrow{\alpha_1} p_1 \cdots$ *in H if and only if there is a run* $\sigma' = p_0 \xrightarrow{\beta_0}_{[\![H]\!]} p_1 \xrightarrow{\beta_1}_{[\![H]\!]} p_2 \cdots$ *in* $[\![H]\!]$*, where* $\beta_i =: t_i \in \mathbb{R}^{\geq 0}$ *iff* $\alpha_i =: (t_i, f_i) \in \mathbb{R}^{\geq 0} \times Ats(Var)$*, and* $\alpha_i = \beta_i$ *otherwise.*

Proof. Both directions follow immediately from the definitions of runs and the timed transition system. Any transition in a run of H also fulfills the definition of a transitions in $[\![H]\!]$. Conversely, any transition in $[\![H]\!]$ implies either a discrete transition in H, or the existence of an activity f_i such that a timed transition exists in H. □

Corollary 5.9. *A state p is reachable in H if and only if it is reachable in* $[\![H]\!]$*.*

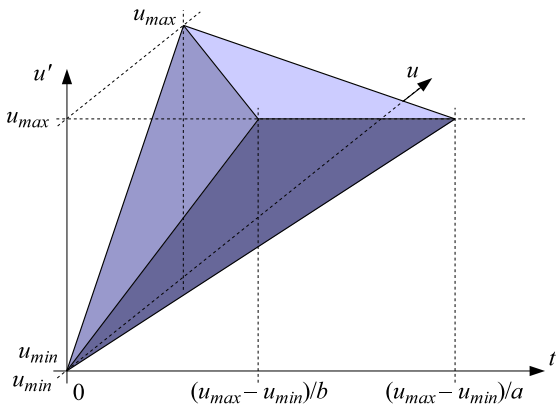


Figure 5.5: Timed transition relation

This abstraction plays an important role in compositional reasoning about hybrid systems and will be discussed in detail in Sect. 8.2. But, as we will show in the next chapter, it does not modify the safety properties of a set of interacting automata if they do not share any variables.

5.4 Related Work

Hybrid systems received increased interest by computer scientists in the beginning of the 1990s, and a number of papers with similar models appeared, with more or less subtle differences in syntax, semantics and terminology. None of these models fit exactly for the purposes of the work in this thesis, so we merge and adapt several of them. In the following we summarize some of the differences and motivate our choices. The relationship between the models in literature is illustrated in Fig. 5.6 with a graph of references between the papers. Some of the papers have appeared in more than one version, so the year of the earliest and the latest publication are noted in brackets. This also explains the circular references between some of them. A side-by-side comparison is shown in Table 5.1. For a very readable introduction to modeling timed and hybrid systems, the reader is referred to [LSW97].

Hybrid Automata The verification of hybrid systems was first discussed by Maler et al. in [MMP92]. While their model of hybrid systems is not explicitly a hybrid automaton, it is in many respects equivalent to the automaton models that we discuss more closely in the following paragraphs. Syntax and terminology of our model are largely taken from [ACH+95], since it unifies major previous work. We add initial states to the definition, which are also present in [ACHH93, HPR94, Hen96], because they play a central role in many technical processes, where a system is usually only behaves safely if it is powered up in certain states, and the model itself is often only valid in a vicinity around behaviors starting in those states. We have transition labels as in [ACH+95], but for the sake of simplicity we omit the stutter transitions, since they have no tangible effect on our work as long as we do not consider shared variables.[5] Consequently, we admit finite as well as infinite runs, as is done in [ACHH93, NOSY92, HPR94, Hen96].

In [ACHH93, HPR94, AHH96, ACH+95], a run is an alternating sequence of time elapse and discrete transitions, which simplifies the notation but requires stutter transitions to allow a sequence of "time elapse, discrete transition, time elapse" as a run (by extending it with a stutter transition), and to change the activity during a time elapse (by inserting a stutter transition between the change).[6] The frameworks in [NOSY92, HPR94] are somewhat counter-intuitive in this sense because they have alternating time elapse and discrete transitions, but no stutter transitions. Similarly, the models in [ACHH93, HPR94, ACH+95] may be counter-intuitive because they do not always have time elapse transitions with duration zero, namely when there is no suitable activity. This may prevent runs with two consecutive discrete transitions and no time elapse in between. In [AHH96, Hen96], zero time elapse is always possible by construction. In the interest of intuitive modeling and a more immediate connection to

[5] We do include sutter transitions in Part III.

[6] Allowing to change the activity is necessary for closure with respect to fusion of runs. Fusion closure asserts that a hybrid automaton is completely determined by the present state of the automaton [AHH96].

the discrete systems of Part I, we choose to allow arbitrary sequences of discrete and timed transitions.

Our definition of parallel composition is similar to that of [ACH+95], but modifies it in two ways. Firstly, extends it to variable sets that can be different, as in [ACHH93]. Secondly, independent transitions must synchronize in [ACH+95] with a stutter transition, to enforce that the controlled variables of the non-participating automaton remain constant. We do not have controlled variables in this part, and no stutter transitions, and therefore treat independent transitions the same as for LTS in Part I.

The continuous dynamics in our model are given by activities, as in [ACHH93, NOSY92, ACH+95], although we do not require conditions such as infinite differentiability or time invariance. In contrast, the activities in [AHH96, Hen96] are specified implicitly by differential equations, which simplifies some arguments, e.g., about approximate equivalence and abstraction.

Linear Hybrid Automata Linear Hybrid Automata were introduced in [ACHH93], but, as in [NOSY92, ACH+95], the dynamics are restricted to piecewise constant derivatives. This restriction is relaxed in [HPR94, AHH96, Hen96] to a linear differential inclusion, which is also the model we use. An extensive treatment of the analysis of linear hybrid automata can be found in [Ho95], which also features a detailed description of methods to over-approximate nonlinear hybrid automata with LHA. A method for constructing conservative LHA models by approximating the linear phase-portrait is given in [HWT96a, HHWT98].

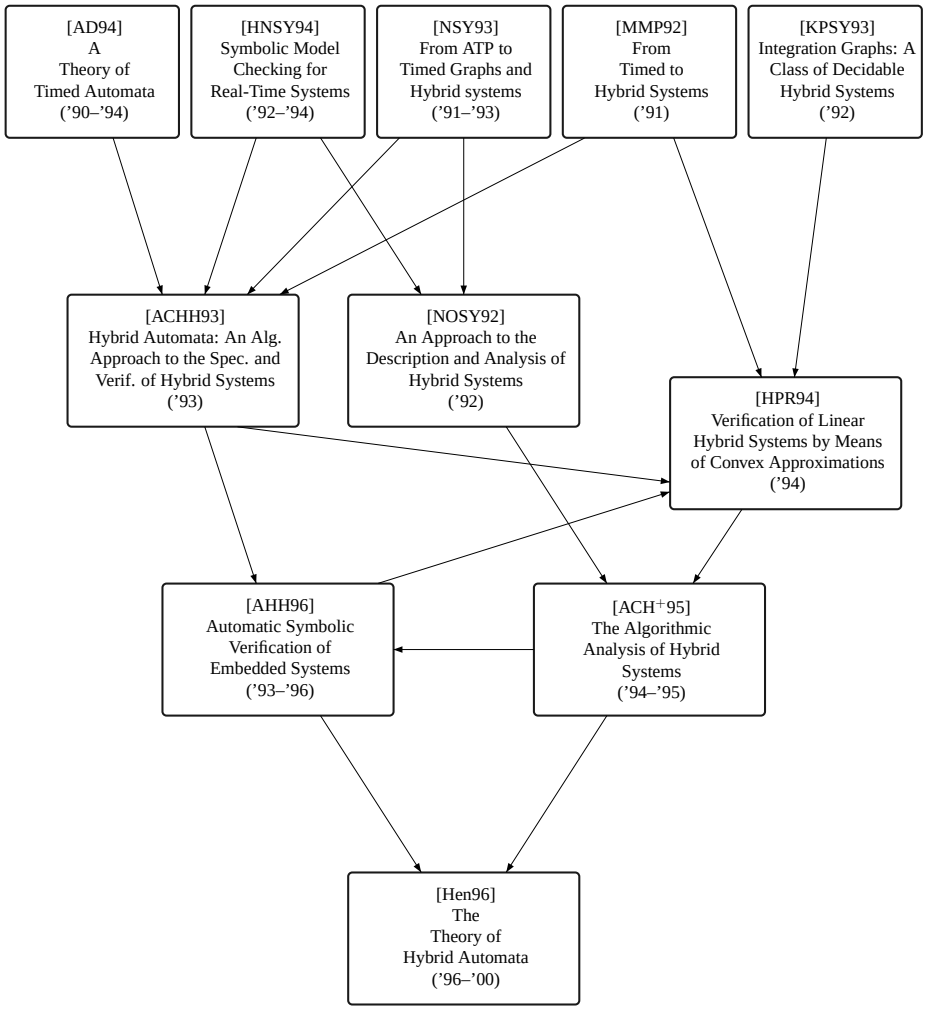


Figure 5.6: Graph of references between hybrid systems definitions. The year of the first and latest version of the publication is shown in brackets

Table 5.1: Side-by-side comparison of hybrid automata definitions

Publication	[ACHH93]	[NOSY92]	[HPR94]	[AHH96]	[ACH+95]	[Hen96]
Invariant	*exception sets* $= \neg Inv$	*time can progress predicate*	invariant sets	convex invariant sets	invariant sets	invariant sets
Initial states	no / yes [a]	no	yes	no	no	yes
Labels	no	yes	no	sets of labels	yes, incl. τ	yes
Transitions	transition relation $\subseteq V \times V$	guard and update function $V \to V$	guard and *action* function $V \to V$	*action predicate* with *closure* operator	transition relation $\subseteq V \times V$	transition relation $\subseteq V \times V$
Stutter trans.	unlabelled	no	no	label set $\emptyset$	label τ	no
Dynamics	*activities* $\mathbb{R}^{\geq 0} \to V \subseteq C^\infty$	time-invariant func. $V \times \mathbb{R}^{\geq 0} \to V$ with $f(v,0) = v$	*activities* $\mathbb{R}^{\geq 0} \to V$, $f(0) = 0$, denoting additive change	convex linear *rate predicate* defining derivative	time-invariant *activities* $\mathbb{R}^{\geq 0} \to V$	*flow* predicate over $X \cup \dot{X}$
Semantics	finite or infinite run (alternating time elapse & discrete transition), piecew. left- or right-closed	LTS (no zero time elapse transitions), run (time elapse & discrete transition or only discrete trans.)	finite or infinite run (alternating time elapse & discrete transition), $v \in Inv(l)$ for $t \in [0, t_i[$	infinite *trajectory* (alternating time elapse & discrete transitions w/o labels)	finite or infinite run (alternating time elapse & discrete transition), LTS	finite or infinite *trajectory* (arbitrary time elapse & discrete transitions), LTS (over invariants)
Zero time elapse	iff $\exists f \in Act(l) : f(0) = v$	never[b]	iff $Act(l) \neq \emptyset$	always	iff $\exists f \in Act(l) : f(0) = v$	always
Parallel composition	cartesian product	–	–	action predicates may change through closure	indep. trans. synchronize with τ-trans., same vars	via LTS, indep. trans. synchronize with 0-time-trans.
LHA deriv.	const.	const.	in linear constraints	in linear predicate	const.	in linear predicate

[a] hybrid system without, hybrid Müller automaton with initial and accepting states
[b] time-can-progress predicate and evolution function admit zero time by definition, but in semantics $t \in \mathbb{R}^{>0}$

Chapter 6

Simulation Relations for Hybrid Automata

In order to be able to compare two automata P and Q, we define a preorder $\preceq$ such that $P \preceq Q$ if any behavior of P finds a match in Q, formally captured by the existence of a *simulation relation* between their states. A state q *simulates* a state p if the system Q shows the same behavior starting from state q as P does starting from state p. In such a comparison, P could be, e.g., an implementation and Q a specification, or P a refined model and Q a more abstract model. Since for safety properties of a hybrid automaton it is sufficient to examine the behavior of its associated LTS, we also define simulation based on the LTSs, following the approach in [Hen96]. For a state q to simulate a state p, an outgoing transition in p must be matched by a transition in q with an identical label. From the TTS-semantics it follows that any time elapse should be matched by an identical time elapse. Depending on the application and the meaning that is attributed to the variables in the process of modeling or when designing the specification, it might be desirable to consider certain variables in the system and specification equivalent. This is imposed by requiring that states in the simulation relation are also in a given equivalence relation [Hen96], formalized by the concept of $\approx$-simulation.

In the following section we define $\approx$-simulation for hybrid automata. In Sect. 6.2, we apply the compositional reasoning framework from Part I to hybrid automata, taking into account the implications for the equivalence relations that accompany $\approx$-simulation. The computation of simulation relations for hybrid automata in $\mathbb{R}^n$ is discussed in Sect. 6.3, and we outline strategies to improve its efficiency, and how to apply parametric analysis. The chapter concludes with a summary of related work in Sect. 6.4.

6.1 Simulation Relations with Equivalence

The simulation between hybrid automata is defined based on the simulation between their corresponding timed transition systems. We use Σ-simulation, since classic sim-

ulation is contained as a special case if the alphabets on both sides of the inequality are equal.

Definition 6.1 ($\approx$-Simulation)**.** *Given LTSs P,Q and an equivalence relation* $\approx \subseteq (S_P \cup S_Q)^2$*,* $Q \approx$**-simulates** P*, written as* $P \preceq_\approx Q$*, if and only if some* $R \subseteq \approx$ *witnesses* $P \preceq_\Sigma Q$*. For hybrid automata* H_1*,* H_2*,* $H_1 \preceq_\approx H_2$ *if and only if* $[\![H_1]\!] \preceq_\approx [\![H_2]\!]$*.*

The equivalence relation $\approx$ is usually defined implicitly, e.g., by demanding that certain variables are identical in P and Q. To simplify the definition of equivalence relations, we write M^C for the reflexive, symmetric and transitive closure of a relation M. In this fairly general definition of simulation with equivalence, two special cases are of particular practical interest:

- ***timed simulation***: all states are considered equivalent so that only the timing between discrete transitions matters,
- ***strict simulation***: a subset of the variables are required to be pairwise identical in both P and Q, while the other variables are irrelevant.

While this distinction is so far not made explicit in literature, timed simulation is used, e.g., in [WL97], and strict simulation, e.g., in [PLS98, TP01, TPL01, TPL02, TP02]. The following example shall illustrate how specifications can require equivalence between some of the variables:

Example 6.1. *Consider a reactor with a constant outflow, a stirrer and a level monitoring controller as shown in Fig. 6.1* [1]*. We use the tank with discrete level sensors* T_s *from Fig. 5.2(c) to model the reactor. The stirrer is not modelled explicitly. The controller, whose model is shown in Fig. 6.2, opens the inlet valve when the level sinks under* x_l *and opens it when the level is above* x_h *with a maximal sampling time of* d_{max}*. It does so based on whether the labels x_high and x_low or their counterparts x_nhigh and x_nlow are enabled. The decision is triggered by a clock d that is restricted by the invariants to a maximal value of* d_{max}*, i.e., the automaton can remain a location a maximal time* d_{max}*, then a transition must be taken. It can equally take the transition at any previous instant. Figure 6.3 shows specification automata for the following properties:*

- *Invariant: The level is always* $x_{min} < x < x_{max}$*.*
- *Sequencing: The stirrer is turned off before the inlet valve is closed.*
- *Timing: The inlet valve is open for a maximum time of* t_{max}*.*

The specification is fulfilled if $T_S||C \preceq_\approx Q_a||Q_b||Q_c$*, with an equivalence relation*

$$\approx = \{((k,u),((l_a,l_b,l_c),v)) \in S_{T_S||C} \times S_{Q_a||Q_b||Q_c} |\ u(x) = v(x)\}^C.$$

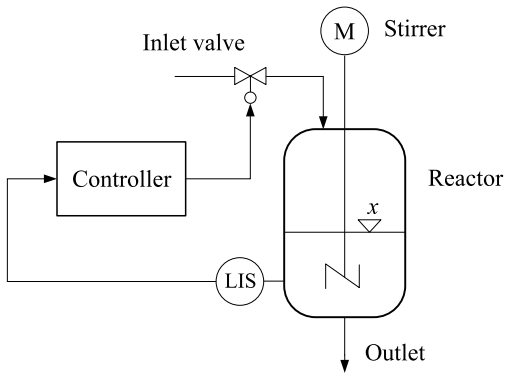


Figure 6.1: Reactor with level monitoring controller

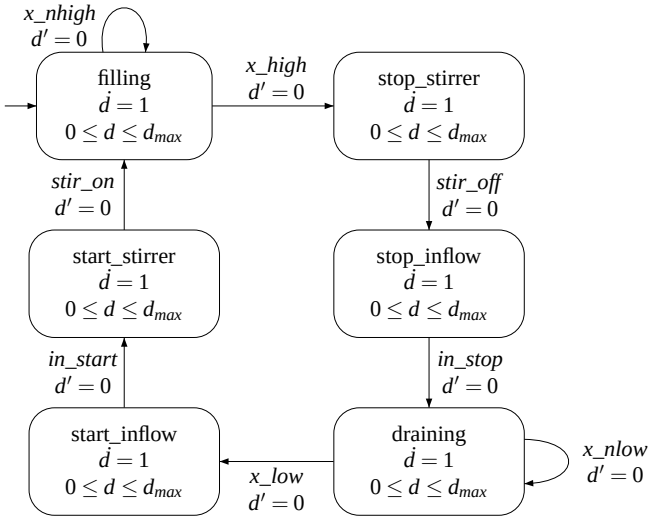
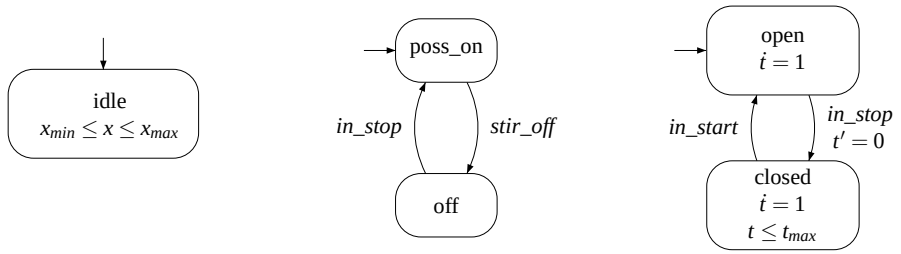


Figure 6.2: Level monitoring controller C



(a) Level invariant Q_a (b) Stirrer off before inlet valve closed Q_b (c) Inlet valve closed at most t_{max}, Q_c

Figure 6.3: Specification automata

Similarity and bisimulation are equivalence relations based on simulation. While similarity defines equivalence based on the branching behavior of the initial states, bisimulation requires a one-to-one equivalence of all states in the relation, i.e., effectively of all reachable states. Definitions and algorithms for similarity and bisimulation of hybrid automata have been given in [Hen96], where they are, amongst other things, applied to obtain finitary equivalence classes.

Definition 6.2 ($\approx$-Similarity)**.** *Given LTS P, Q and an equivalence relation $\approx \subseteq (S_P \cup S_Q)^2$, Q is $\approx$-**similar** to P, written as $P \simeq_\approx Q$, if and only if $P \preceq_\approx Q$ and $Q \preceq_\approx P$. For hybrid automata H_1, H_2, $H_1 \simeq_\approx H_2$ if and only if $[\![H_1]\!] \simeq_\approx [\![H_2]\!]$.*

Definition 6.3 ($\approx$-Bisimulation)**.** *Given LTS P, Q and an equivalence relation $\approx \subseteq (S_P \cup S_Q)^2$, a relation $R_\simeq \subseteq S_P \times S_Q$ is a $\approx$-**bisimulation relation** if and only if $R_\simeq$ witnesses $P \preceq_\approx Q$ and $R_\simeq^{-1}$ witnesses $Q \preceq_\approx P$. Q is $\approx$-**bisimilar** to P, written as $P \cong_\approx Q$, if and only if there exists such a relation. For hybrid automata H_1, H_2, $H_1 \cong_\approx H_2$ if and only if $[\![H_1]\!] \cong_\approx [\![H_2]\!]$.*

As established in Prop. 3.3, simulation is a necessary condition for similarity, and similarity is necessary for bisimulation. It immediately follows that the equivalence is also preserved:

Proposition 6.4. *For all P, Q, $P \cong_\approx Q \Rightarrow P \simeq_\approx Q \Rightarrow P \preceq_\approx Q$.*

We will use bisimulation and Prop. 6.4 extensively in the next sections in order to commute the parallel composition operator and the TTS-semantics. This is the key to applying the compositional framework of Part I, and only possible because we exclude shared variables.

[1]The level monitoring controller is a popular example in hybrid systems literature. Similar models, although not quite identical, can be found in [Hoo93, HPR94], the latter of which appears again in [ACH+95].

6.2 Compositional Reasoning

For hybrid automata with disjoint variables, the timed transition system and the parallel composition operator are commutative (up to structural isomorphism) if the equivalence relations fulfill certain conditions. Together with invariance under composition, this property allows us to carry over the compositional proofs of Part I from labelled transition systems to hybrid automata. This is formalized by the following propositions.

Lemma 6.5. *For any valuations v_1, v_2 over disjunct sets of variables Var_1, respectively Var_2, there exists a unique valuation v over $Var = Var_1 \cup Var_2$ with $v{\downarrow}_{Var_i} = v_i$ for $i = 1,2$. Given any v it holds that v_1 and v_2 exist and are unique.*

Proof. Let v be defined as $v(x) = v_i(x)$ for all $x \in Var_i$, $i = 1,2$. Since the Var_i partition and cover Var, v exists and is unique. □

Proposition 6.6. *For any H_1, H_2 with disjoint variables and an equivalence relation $\approx$ such that $((l_1,l_2),v) \approx ((l_1,v_1),(l_2,v_2))$ for $v{\downarrow}_{Var_{H_i}} = v_i$ for $i = 1,2$, it holds that $[\![H_1||H_2]\!] \cong_{\approx} [\![H_1]\!]||[\![H_2]\!]$.*

Proof. First we show that $[\![H_1||H_2]\!] \preceq_{\approx} [\![H_1]\!]||[\![H_2]\!]$. Let

$$R = \left\{ (((l_1,l_2),v),((l_1,v_1),(l_2,v_2))) \,\middle|\, v{\downarrow}_{Var_{H_i}} = v_i \text{ for } i = 1,2 \right\}.$$

With Lemma 6.5 and the definition of parallel composition it follows that for all initial states $((l_1,l_2),v) \in Init_{[\![H_1||H_2]\!]}$ there exists some $((l_1,v_1),(l_2,v_2)) \in Init_{[\![H_1]\!]||[\![H_2]\!]}$. Furthermore, $R \subseteq \approx$ and the alphabets on both sides are equal, so that only condition (i) of Σ-simulation applies. From the definition of parallel composition of hybrid automata it follows that the projections of the invariants, activities and discrete transition relations of $H_1||H_2$ exist in H_1 and H_2. Therefore for each transition from a state $((l_1,l_2),v)$ there exists a corresponding transition from the state $((l_1,v_1),(l_2,v_2))$, $v{\downarrow}_{Var_{H_i}} = v_i$ for $i = 1,2$, and their target states are again in R. Consequently, R is a witness for $[\![H_1||H_2]\!] \preceq_{\approx} [\![H_1]\!]||[\![H_2]\!]$. A similar argument shows that R^{-1} is a witness for $[\![H_1]\!]||[\![H_2]\!] \preceq_{\approx} [\![H_1||H_2]\!]$. □

For hybrid automata with disjunct variables, $\approx$-simulation is a precongruence with respect to parallel composition if it is consistent with the equivalence relations used. Since an equivalence relation is reflexive by definition, it remains to discuss transitivity and invariance under composition:

Proposition 6.7 (Transitivity)**.** *For any H_1, H_2, H_3 with disjunct variables, $H_1 \preceq_{\approx_1} H_2$ and $H_2 \preceq_{\approx_2} H_3$ imply $H_1 \preceq_{\approx_3} H_3$ if $(l_1,v_1) \approx_1 (l_2,v_2)$ and $(l_2,v_2) \approx_2 (l_3,v_3)$ imply $(l_1,v_1) \approx_3 (l_3,v_3)$.* [2]

[2] The transitivity of the equivalence relations within themselves does not help here, since they range over different state spaces.

Proof. Let R_1 be a witness for $[\![H_1]\!] \preceq_{\approx_1} [\![H_2]\!]$, and R_2 for $[\![H_2]\!] \preceq_{\approx_2} [\![H_3]\!]$. By Def. 6.1, $R_1 \subseteq \approx_1$ and $R_2 \subseteq \approx_2$. According to Prop. 3.11, $R = \{(p,s) | \exists q : (p,q) \in R_1 \wedge (q,s) \in R_2\}$ is a witness for $[\![H_1]\!] \preceq_\Sigma [\![H_3]\!]$. With the hypothesis it follows that $R \subseteq \approx_3$, so R is a witness for $H_1 \preceq_{\approx_3} H_3$. □

Proposition 6.8 (Invariance under Composition)**.** *For any H_1, H_2, H_3, with $Var_{H_i} \cap Var_{H_3} = \emptyset$ for $i = 1,2$, $H_1 \preceq_{\approx_1} H_2$ implies $H_1||H_3 \preceq_{\approx_2} H_2||H_3$ if $(l_1,v_1) \approx_1 (l_2,v_2)$ implies $((l_1,l_3),u_1) \approx_2 ((l_2,l_3),u_2)$ with $u_i{\downarrow}_{Var_{H_i}} = v_i$ for $i = 1,2$ and $u_1{\downarrow}_{Var_{H_3}} = u_2{\downarrow}_{Var_{H_3}}$.*

Proof. Let R_1 be a witness for $[\![H_1]\!] \preceq_{\approx_1} [\![H_2]\!]$. By Def. 6.1, $R_1 \subseteq \approx_1$. According to Prop. 3.11, $R = \{(((l_1,v_1),(l_3,v_3)),((l_2,v_2),(l_3,v_3))) | \ ((l_1,v_1),(l_2,v_2)) \in R_1\}$ is a witness for $[\![H_1]\!]||[\![H_3]\!] \preceq_\Sigma [\![H_2]\!]||[\![H_3]\!]$. With Prop. 6.6 and Prop. 6.4 it follows that

$$R' = \left\{ (((l_1,l_3),v),((l_1,v_1),(l_3,v_3))) \middle| v{\downarrow}_{Var_{H_i}} = v_i \text{ for } i = 1,3 \right\}$$

is a witness for $[\![H_1||H_3]\!] \preceq_\Sigma [\![H_1]\!]||[\![H_3]\!]$, and

$$R'' = \left\{ (((l_2,v_2),(l_3,v_3)),((l_2,l_3),v)) \middle| v{\downarrow}_{Var_{H_i}} = v_i \text{ for } i = 2,3 \right\}$$

is a witness for $[\![H_2]\!]||[\![H_3]\!] \preceq_\Sigma [\![H_2||H_3]\!]$. With transitivity of Prop. 3.11 follows that

$$R''' = \left\{ (((l_1,l_3),v),((l_2,v_2),(l_3,v_3))) \middle| \ ((l_1,v{\downarrow}_{Var_{H_1}}),(l_2,v_2)) \in R_1, v{\downarrow}_{Var_{H_3}} = v_3 \right\}$$

witnesses $[\![H_1||H_3]\!] \preceq_\Sigma [\![H_2]\!]||[\![H_3]\!]$, and again by transitivity

$$R'''' = \{(((l_1,l_3),u_1),((l_2,l_3),u_2))) | \ ((l_1,v_1),(l_2,v_2)) \in R_1, \\ u_i{\downarrow}_{Var_{H_i}} = v_i \text{ for } i = 1,2, u_1{\downarrow}_{Var_{H_3}} = u_2{\downarrow}_{Var_{H_3}} \Big\}.$$

is a witness for $[\![H_1||H_3]\!] \preceq_\Sigma [\![H_2||H_3]\!]$ Under the hypothesis it follows that $R \subseteq \approx_2$, so R'''' is a witness for $H_1 \preceq_{\approx_2} H_3$. □

We now formulate two main theorems in our compositional reasoning framework: compositionality and decomposition of the specification. As before, we use the witnessing simulation relations from corresponding proofs in Part I to obtain criteria for the equivalence relations.

Theorem 6.9 (Compositionality)**.** *Consider HA P_1, P_2, Q_1, Q_2, with $Var_{P_i} \cap Var_{Q_i} = \emptyset$ for $i = 1,2$. If $P_i \preceq_{\approx_i} Q_i$ for $i = 1,2$, then $P_1||P_2 \preceq_\approx Q_1||Q_2$ holds if $((k_i,u_i),(l_i,v_i)) \in \approx_i$ for $i = 1,2$ implies $(((k_1,k_2),u),((l_1,l_2),v)) \in \approx$, $u{\downarrow}_{Var_{P_i}} = u_i$, $v{\downarrow}_{Var_{Q_i}} = v_i$.*

Proof. According to Prop. 3.7, compositionality follows from the preorder properties, commutativity of the composition operator and invariance under composition. Let R_i be witnessing relations for $P_i \preceq_{\approx_i} Q_i$. Let $p_i = (k_i,u_i)$ and $q_i = (l_i,v_i)$. With

Prop. 3.11, it follows from $[\![P_1]\!] \preceq_\Sigma [\![Q_1]\!]$ that $[\![P_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_1]\!]||[\![P_2]\!]$, with a witness $R' = \{((p_1,p_2),(q_1,p_2))|(p_1,q_1) \in R_1\}$. Similarly, it follows from $[\![P_2]\!] \preceq_\Sigma [\![Q_2]\!]$ and commutativity that a witness for $[\![Q_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_1]\!]||[\![Q_2]\!]$ is given by $R'' = \{((q_1,p_2),(q_1,q_2))|(p_2,q_2) \in R_2\}$. Transitivity then implies $[\![P_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_1]\!]||[\![Q_2]\!]$ with a witness $R''' = \{((p_1,p_2),(q_1,q_2))|(p_1,q_1) \in R_1 \wedge (p_2,q_2) \in R_2\}$. Applying Prop. 6.6, it finally holds that $[\![P_1||P_2]\!] \preceq_\Sigma [\![Q_1||Q_2]\!]$ with a witness

$$R = \{(((k_1,k_2),u),((l_1,l_2),v))|((k_i,u{\downarrow}_{Var_{P_i}}),(l_i,v{\downarrow}_{Var_{Q_i}})) \in R_i \text{ for } i = 1,2\}$$

Under the hypothesis it holds that $R \subseteq \approx$, so that R witnesses $P_1||P_2 \preceq_\approx Q_1||Q_2$. □

The following example shall illustrate how compositionality justifies the use of abstractions in proofs:

Example 6.2. *Consider the tank level monitor system of Ex. 6.1 with the non-linear tank model T from Fig. 5.1(b), the level sensors in Figs. 5.2(a) and 5.2(b), and the controller C in Fig. 6.2. The system is the composition $P = T||L_l||L_h||C$. To perform an algorithmic analysis, we can use an overapproximation by a linear hybrid automaton, e.g., $\hat{T}$ from Ex. 5.3, shown in Fig. 5.3. The models for the level sensors remain the same. The validity of the abstraction, i.e., that $T \preceq_{\approx_T} \hat{T}$ holds with $\approx_T = \{((k,u),(l,v)) \in S_T \times S_{\hat{T}} |\ u(x) = v(x)\}^C$ is usually guaranteed if one of the well-established methods was used, otherwise it can be established manually. Then compositionality yields $T||L_l||L_h||C \preceq_\approx \hat{T}||L_l||L_h||C$, with*

$$\approx = \{((k,u),(l,v)) \in S_{T||L_l||L_h||C} \times S_{\hat{T}||L_l||L_h||C} |\ u(x) = v(x)\}^C.$$

So instead of P we can verify $\hat{P} = \hat{T}||L_l||L_h||C$, and any safety properties will be preserved.

The second main theorem in the compositional framework states that the verification of a decomposed specification, i.e., the right side of the inequality $P \preceq Q$, is equivalent to verifying the composed specification. Therefore, the parallel composition operator behaves like a logical AND operator:

Theorem 6.10 (Decomposition of Specification)**.** *Consider any HA P, Q_1, Q_2, with $Var_{Q_1} \cap Var_{Q_2} = \emptyset$. If $P \preceq_\approx Q_1||Q_2$ holds, then $P \preceq_{\approx_i} Q_i$ if $(p,((l_1,l_2),v)) \in \approx$ implies $(p,(l_i,v{\downarrow}_{Var_{Q_i}})) \in \approx_i$ for $i = 1,2$. If $P \preceq_{\approx_i} Q_i$ for $i = 1,2$, then $P \preceq_\approx Q_1||Q_2$ holds if $(p,(l_i,v_i)) \in \approx_i$ for $i = 1,2$ implies $(p,((l_1,l_2),v)) \in \approx$, $v{\downarrow}_{Var_{Q_i}} = v_i$.*

Proof. We follow the proof structure of Theorem 3.20 and construct a witnessing simulation relation to be able to reason about the equivalence relations. Assume that $P \preceq_\approx Q_1||Q_2$ holds, witnessed by a relation R. With Prop. 6.6, Prop. 6.4 and transitivity it follows that $[\![P]\!] \preceq_\Sigma [\![Q_1]\!]||[\![Q_2]\!]$ with a witnessing simulation relation $R' = \{(p,((l_1,v_1),(l_2,v_2)))|(p,((l_1,l_2),v)) \in R, v{\downarrow}_{Var_{Q_i}} = v_i \text{ for } i = 1,2\}$. According to Prop. 3.18, $R'_i = \{((q_1,q_2),q_i)\}$ are witnesses for $[\![Q_1]\!]||[\![Q_2]\!] \preceq_\Sigma [\![Q_i]\!]$, $i = 1,2$.

From transitivity follows the sufficient direction of the theorem with witnesses $R_i'' = \{(p,(l_i,v_i)) | \exists l_j, v : (p,((l_1,l_2),v)) \in R, v{\downarrow}_{Var_{Q_i}} = v_i\}$ for $(i,j) \in \{(1,2),(2,1)\}$. Under the hypothesis, $R_i'' \subseteq \approx_i$, which concludes this part of the proof.

For the necessary direction, assume that $P \preceq_{\approx} Q_i$ with witnessing R_i for $i = 1,2$. Using compositionality and Prop. 3.19, we get $[\![P]\!] \preceq_{\Sigma} [\![Q_1]\!] || [\![Q_2]\!]$ with a witnessing relation $R' = \{(p,(q_1,q_2)) | (p,q_1) \in R_1 \wedge (p,q_2) \in R_2\}$. With Prop. 6.6, Prop. 6.4 and transitivity it follows that $R = \{(p,((l_1,l_2),v)) | (p,(l_i,v{\downarrow}_{Var_{Q_i}})) \in R_i$ for $i = 1,2\}$ is witnessing $P \preceq_{\Sigma} Q_1 || Q_2$, and under the hypothesis holds $R \subseteq \approx$. □

The following example shall illustrate the usefullness of decomposing the specification, while taking into account equivalence relations:

Example 6.3. *Consider the tank level monitoring system of Ex. 6.1 and its specification $Q = Q_a || Q_b || Q_c$ shown in Fig. 6.3. Let the equivalence relations for the decomposition be given by*

$$\begin{aligned}
\approx_a &= \{((k,u),(l_a,v)) \in S_{T_S||C} \times S_{Q_a} |\ u(x) = v(x)\}^C \\
\approx_b &= \{((k,u),(l_b,v)) \in S_{T_S||C} \times S_{Q_b}\}^C \\
\approx_c &= \{((k,u),(l_c,v)) \in S_{T_S||C} \times S_{Q_c}\}^C \\
\approx &= \{((k,u),((l_a,l_b,l_c),v)) \in S_{T_S||C} \times S_{Q_a||Q_b||Q_c} |\ u(x) = v(x)\}^C
\end{aligned}$$

With Theorem 6.10, we can the decompose the specification and to do show that $T_S||C \preceq_{\approx} Q_a||Q_b||Q_c$. The application of Theorem 6.10 allows to verify each specification separately in the form

$$\frac{\begin{array}{c} T_S||C \preceq_{\approx_a} Q_a \\ T_S||C \preceq_{\approx_b} Q_b \\ T_S||C \preceq_{\approx_c} Q_c \end{array}}{T_S||C \preceq_{\approx} Q_a||Q_b||Q_c}.$$

Using the equivalence relation

$$\approx_{ab} = \{((k,u),(l_a,l_b),v)) \in S_{T_S||C} \times S_{Q_a||Q_b} |\ u(x) = v(x)\}^C,$$

this is easily shown by splitting the proof in two steps:

$$\frac{\begin{array}{c} T_S||C \preceq_{\approx_a} Q_a \\ T_S||C \preceq_{\approx_b} Q_b \end{array}}{T_S||C \preceq_{\approx_{ab}} Q_a||Q_b}, \qquad \frac{\begin{array}{c} T_S||C \preceq_{\approx_{ab}} Q_a||Q_b \\ T_S||C \preceq_{\approx_c} Q_c \end{array}}{T_S||C \preceq_{\approx} Q_a||Q_b||Q_c}.$$

It is easy to see that the decomposed specifications in the above example each only refer to certain aspects of the system. We can further simplify the proof by using compositionality and abstraction:

Example 6.4. *In Ex. 6.3, we showed that the specification $Q = Q_a||Q_b||Q_c$ of the tank level monitoring system of Ex. 6.1 can be verified by checking Q_a, Q_b and Q_c separately. The specification Q_b refers only to controller commands, so it can be verified using only the controller C. Let*

$$\approx_b' \;=\; \{((k,u),(l_b,v)) \in S_C \times S_{Q_b}\}^C$$

It is easy to see that a simulation relation that witnesses $C \preceq_{\approx_b'} Q_b$ is given by

$$R = \{((k,l),(l,v))|(k \in \{\text{filling}, \text{stop_stirrer}, \text{draining}, \text{start_inflow}, \text{start_stirrer}\} \\ \wedge\, l = \text{poss_on}) \vee (k = \text{stop_inflow} \wedge l = \text{on})\ \}.$$

With invariance under composition according to Theorem 6.8 and decomposition of the specification follows:

$$\frac{C \preceq_{\approx_b'} Q_b}{T_S||C \preceq_{\approx_b} Q_b}.$$

For Q_a and Q_c, the stirrer operation is irrelevant. We construct an abstraction of the controller that omits these steps, which is shown in Fig. 6.4. Since we do not want to model the resets of the clock after the stirrer commands, we use a different clock and for clarity call it δ instead of d. The abstracted model adds the delay in the stirrer commands to the delay the states that follow, resulting in a delay of $2d_{max}$. We impose no particular equivalence between the old and the new clock, so let

$$\approx_C \;=\; \{((k,u),(l,v)) \in S_C \times S_{\hat{C}}\}^C.$$

It is easy to see that $C \preceq_{\approx_C} \hat{C}$, or it can be verified algorithmically. In addition, consider the abstracted tank model of Ex. 6.2. Using the equivalence relation

$$\approx' \;=\; \{((k,u),(l,v)) \in S_{T||C} \times S_{\hat{T}||\hat{C}}\}^C,$$

we get with compositionality that

$$\frac{\begin{array}{c} T \preceq_{\approx_T} \hat{T} \\ C \preceq_{\approx_C} \hat{C} \end{array}}{T||C \preceq_{\approx'} \hat{T}||\hat{C}}.$$

The abstracted system $\hat{P} = \hat{T}||L_l||L_h||\hat{C}$ can therefore be used to verify Q_a and Q_c. With equivalence relations

$$\approx_a' \;=\; \{((k,u),(l,v)) \in S_{\hat{T}||L_l||L_h||\hat{C}} \times S_{Q_a} |\ u(x) = v(x)\}^C,$$
$$\approx_c' \;=\; \{((k,u),(l,v)) \in S_{\hat{T}||L_l||L_h||\hat{C}} \times S_{Q_a} |\ u(x) = v(x)\}^C,$$

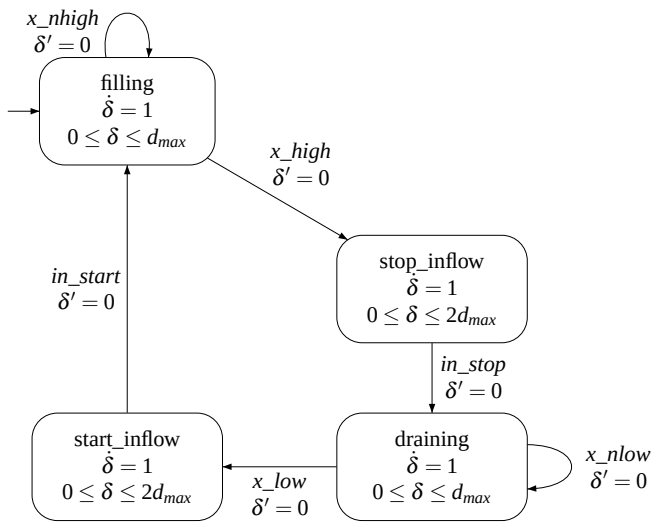


Figure 6.4: Abstraction of level monitoring controller $\hat{C}$ after omitting stirrer commands

the complete proof now goes as follows:

$$\frac{\begin{array}{c} C \preceq_{\approx_b'} Q_b \\ T \preceq_{\approx_T} \hat{T} \\ C \preceq_{\approx_C} \hat{C} \\ \hat{T}||L_l||L_h||\hat{C} \preceq_{\approx_a'} Q_a \\ \hat{T}||L_l||L_h||\hat{C} \preceq_{\approx_c'} Q_c \end{array}}{T||L_l||L_h||C \preceq_{\approx} Q_a||Q_b||Q_c}.$$

Since the worst case complexity of verifying simulation increases exponentially with the number of systems involved, we can expect the collection of less and simpler modules to be considerably quicker than the verification of the full system with the full specification.

Abstracted models, such as in the examples above, can help to considerably speed up the analysis, and experimental results will be provided in Part 11. In the next section, we will discuss methods to compute simulation relations algorithmically, based on geometric operations in $\mathbb{R}^n$.

6.3 Computing Simulation Relations in $\mathbb{R}^n$

The computation of a simulation relation R that witnesses $P \preceq Q$ for hybrid automata P and Q must be done in a symbolic fashion, i.e., with sets of states at a time, in order to obtain finite algorithm, or at least a semi-algorithm. In view of an implementation, valuations over n variables are interpreted as points in $\mathbb{R}^n$, and a relation R as a map from pairs of locations to $\mathbb{R}^{n+m}$, where n and m are the number of variables in P and Q. Where useful, valuation identifiers will be written underneath the set to clarify the ordering of variables. While the computations as such are not different from their discrete versions in Sect. 3.3, their version in $\mathbb{R}^n$ requires embedding and reordering operations that add to the computational cost. The symbolic versions illustrate where the computationally expensive difference operation is necessary. The following operations are used:

- intersection: $\underset{u}{A} \cap \underset{u}{B} = \{u | u \in A \wedge u \in B\}$,
- difference: $\underset{u}{A} \cap \neg \underset{u}{B} = \{u | u \in A \wedge u \notin B\}$,
- projection: $\underset{u,v}{A}\downarrow_u = \{u | \exists v : (u,v) \in A\}$,
- embedding: $\underset{u}{A}|^{u,v} = \{(u,v) | u \in A\}$,
- reordering: $\underset{u,v}{A}|^{v,u} = \{(v,u) | (u,v) \in A\}$.

To fulfill $P \preceq Q$, a state of a hybrid automaton P must conform with Q in discrete and timed transitions:

- a discrete transition is either not enabled in P or is matched in Q and
- a time-elapse is possible in P must also be possible in Q.

If it fails to do so, it is called a ***bad*** state. The simulation relation is computed by successive approximation. First, it is initialized with the states in the equivalence relation $\approx$. Then bad states are subtracted until convergence. Just as with reachability, this is undecidable for linear hybrid automata but does converge in many practical cases. To force convergence, the relation can be restricted by widening the complement $\neg R$.

Let k be a location of P, l a location of Q and $B^{tr}(k,l)$ the set of states in P that have no matching discrete transitions in Q. Similarly, let $B^{te}(k,l)$ be those that have no matching timed transitions in Q. Then R is the largest fixed-point of the operator

$$R(k,l) := R(k,l) \cap \neg B^{tr}(k,l) \cap \neg B^{te}(k,l). \tag{6.1}$$

Finally, Q simulates P if all initial states in P find a match in the simulation relation R, i.e., if for all locations k in P holds:

$$\underset{u}{Init_P}(k) \cap \neg \bigcup_l \left(\underset{v}{Init_Q}(l)|^{u,v} \cap \underset{u,v}{R}(k,l) \right) \Bigg|_{\downarrow u} = \emptyset. \tag{6.2}$$

procedure *GetSimRel*
Input: Hybrid automata P,Q, equivalence relation R_E over $S_P \cup S_Q$
Output: a simulation relation R
$R := \{(p,q) \in S_P \times S_Q | (p,q) \in R_E\}$
while there exist (k,l) **with** $R(k,l) \cap (B^{tr}(k,l) \cup B^{te}(k,l)) \neq \emptyset$ **do**
$R(k,l) := R(k,l) \cap \neg B^{tr}(k,l) \cap \neg B^{te}(k,l)$
end while

Figure 6.5: Semi-algorithm for computing a simulation relation

procedure *HAGetBisimRel*
Input: HA P,Q, equivalence relation R_E
Output: a bisimulation relation R

$R := \{(p,q) \in S_P \times S_Q | (p,q) \in R_E\}$
$R' := \emptyset$
while $R \neq R'$ **do**
$R' := R$
for all $(k,l) \in Loc_P \times Loc_Q$ **do**
$R(k,l) := R(k,l) \cap \neg B^{tr}(k,l) \cap \neg B^{te}(k,l)$
$R(l,k) := R(l,k) \cap \neg B^{tr}(k,l)^{-1} \cap \neg B^{te}(k,l)^{-1}$
end for
end while

Figure 6.6: Semi-algorithm for computing a bisimulation relation

The symbolic version of the discrete algorithm from Fig. 3.4 is shown in Fig. 6.5. The algorithms for finding a simulation relation are readily adapted to generate a bisimulation by initializing R symmetrically and subtracting bad states B in $R(k,l)$ not only from $R(k,l)$, but also subtracting B^{-1} from $R(l,k)$. A simple semi-algorithm is shown in Fig. 6.6.

6.3.1 Symbolic Computation of Bad States

In the definition of simulation, there is no distinction between discrete and timed transitions. In both cases, a transition in P must be matched by a transition in Q if the label is in both alphabets. We differentiate between them in the following because they are defined by different sets of the hybrid automaton. Since the labels of a timed transition are always in the alphabet of both TTSs, we only need to take into account the case Def. 3.9(i).

Discrete Transitions With the definition of the TTS, we can associate a transition with label $a \in Lab_P$ with a discrete transition in the hybrid automaton P. According to

Def. 3.9(i), a state (l,v) simulates a state (k,u) in a discrete transition if

$$(k,u) \xrightarrow{a,\mu}_P (k',u') \Rightarrow \exists (l',v') : [(l,v) \xrightarrow{a,\eta}_Q (l',v') \wedge (u',v') \in R(k',l')].$$

Consider a pair of transitions, s in P and t in Q. With the definition of the timed transition system, the set of states that violate this condition for are:

$$\begin{aligned} B_{(i)}(s,t) = \neg \big\{ (u,v) | \forall u' : \neg [u \in Inv(k) \wedge u' \in Inv(k') \wedge (u,u') \in \mu] \\ \vee \exists v' : [v \in Inv(l) \wedge v' \in Inv(l') \wedge (v,v') \in \eta \wedge (u',v') \in R(k',l')] \big\} \end{aligned}$$

For a symbolic computation with the projection operator, it must be transformed to have only existential quantifiers: [3]

$$\begin{aligned} B_{(i)}(s,t) = \neg\neg \{ (u,v) | \exists u' : [u \in Inv(k) \wedge u' \in Inv(k') \wedge (u,u') \in \mu] \\ \wedge \nexists v' : [v \in Inv(l) \wedge v' \in Inv(l') \wedge (v,v') \in \eta \wedge (u',v') \in R(k',l')] \} \end{aligned}$$

The symbolic computation is carried out in two projections, corresponding to first the quantification of v', and then the quantification over u'. We will now describe the bad states using the symbolic operations. The enabled states in P are

$$\underset{u,u'}{E_P}(s) = \underset{u}{Inv}(k)|^{u,u'} \cap \underset{u'}{Inv}(k')|^{u,u'} \cap \underset{u,u'}{\mu}, \tag{6.3}$$

the enabled states in Q with $(u',v') \in R(k',l')$ are

$$\underset{u',v}{E_Q}(t) = \left(\underset{v}{Inv}(l)|^{u',v,v'} \cap \underset{v'}{Inv}(l')|^{u',v,v'} \cap \underset{v,v'}{\eta}|^{u',v,v'} \cap \underset{u',v'}{R}(k',l')|^{u',v,v'} \right) \Big\downarrow_{u',v}. \tag{6.4}$$

Then the bad states for the pair of transitions s and t are given by

$$B^{tr}_{(i)}(s,t) = \left(\underset{u,u'}{E_P}(s)|^{u,u',v} \cap \neg \underset{u',v}{E_Q}(t)|^{u,u',v} \right) \Big\downarrow_{u,v} \tag{6.5}$$

A state in a pair of locations (k,l) is only bad if it is bad for all pairs of transitions s and t with the same label $a \in \Sigma_P \cap \Sigma_Q$:

$$B^{tr}_{(i)}(k,l,a) = \bigcup_{s=(k,a,\mu,k')} \left(\bigcap_{t=(l,a,\eta,l')} B^{tr}_{(i)}(s,t) \right). \tag{6.6}$$

A similar argument yields the states that violate condition (ii) for a label $b \in \Sigma_Q \backslash \Sigma_P$:

$$B^{tr}_{(ii)}(k,l,b) = \bigcup_{t=(l,b,\eta,l')} \left(\neg \underset{v,v'}{\eta}|^{u',v,v'} \cup \neg \underset{u',v'}{R}(k',l')|^{u',v,v'} \right) \Big\downarrow_{u,v}. \tag{6.7}$$

[3] For a set $A \subseteq \{(u,v)\}$, the projection eliminates the existential quantifier: $A\downarrow_u = \{u \in A | \exists v : A(u,v)\}$. The complement of this quantifier is $\neg(A\downarrow_u) = \{u \in A | \nexists v : A(u,v)\} = \{u \in A | \forall v : \neg A(u,v)\}$.

For condition (iii) and a label $c \in \Sigma_P \backslash \Sigma_Q$, the bad states are given by

$$B^{tr}_{(iii)}(k,l,c) = \bigcup_{t=(l,c,\eta,l')} \left(\underset{u,u'}{\mu} |^{u,u',v'} \cap \neg \underset{u',v'}{R}(k',l')|^{u,u',v'} \right) \Big|_{\downarrow u,v} . \tag{6.8}$$

In total, the set of bad states due to discrete transitions is the union of the above sets:

$$B^{tr}(k,l) = \bigcup_{a \in \Sigma_P \cap \Sigma_Q} B^{tr}_{(i)}(k,l,a) \cup \bigcup_{b \in \Sigma_Q \backslash \Sigma_P} B^{tr}_{(ii)}(k,l,b) \cup \bigcup_{c \in \Sigma_P \backslash \Sigma_Q} B^{tr}_{(iii)}(k,l,c). \tag{6.9}$$

Timed Transitions Since $\mathbb{R}^{\geq 0}$ is in the alphabet of both $[\![P]\!]$ and $[\![Q]\!]$, only condition Def. 3.9(i) applies. A state (k,u) simulates a state (l,u) in a timed transition if

$$(k,u) \xrightarrow{t}_{[\![P]\!]} (k',u') \Rightarrow \exists (l',v') : [(l,v) \xrightarrow{t}_{[\![Q]\!]} (l',v') \wedge (u',v') \in R(k',l')].$$

For an automaton P in a location k, the set of time successors in the space of (u,u',t) is given by the timed transition relation $\rightsquigarrow_P (k)$ of the timed transition system, as defined in Sect. 5.3. Using this notation, we get the timed bad states in (k,l):

$$B^{te}(k,l) = \neg\{(u,v) | \forall t \geq 0 : \forall u', (u,u',t) \in \rightsquigarrow_P (k) : \exists v', (v,v',t) \in \rightsquigarrow_Q (l) \wedge R(u',v')\}$$

Using only existential quantifiers, the bad states are given by

$$\begin{aligned} B^{te}(k,l) &= \neg\{(u,v) | \nexists t \geq 0 : \neg[\nexists u', (u,u',t) \in \rightsquigarrow_P (k) : \\ &\qquad \neg[\exists (v,v',t) \in \rightsquigarrow_Q (l) \wedge R(u',v')]]\} \\ &= \neg\neg\{(u,v) | \exists t \geq 0 : [\exists u', (u,u',t) \in \rightsquigarrow_P (k) : \neg[\exists (v,v',t) \in \rightsquigarrow_Q (l) \wedge R(u',v')]].\} \end{aligned}$$

Symbolically, the inner bracket can be written as

$$\underset{u',v,t}{E^{te}_Q}(k,l) = \left(\underset{v,v',t}{\rightsquigarrow_Q} (l) \, |^{u',v,v',t} \cap \underset{u',v'}{R} |^{u',v,v',t} \right) \Big|_{\downarrow u',v,t} \tag{6.10}$$

and the timed bad states for a pair of locations (k,l) are then given by

$$B^{te}(k,l) = \left(\underset{u,u',t}{\rightsquigarrow_p} (k) |^{u,u',v,t} \cap \neg \underset{u',v,t}{E^{te}_Q}(k,l) |^{u,u',v,t} \right) \Big|_{\downarrow u,v} . \tag{6.11}$$

The computation of R is complicated by two nested difference operations, the main difference operation in (6.1) and the ones in (6.5), (6.7), (6.8) and (6.11). In the general case, these can not be simplified because the projection operations are not commutative. It is therefore essential to the performance of any implementation to emphasize the simplification of these operations, as discussed in the next section.

6.3.2 Performance Improvements

The following ways to improve the performance of the computation of a simulation relation have been reported in literature. Experimental results will be presented in Part 11.

Simplification of the Simulation Relation: According to (6.1)-(6.11) two nested difference operations are necessary. Given the complexity of the difference operation, this is a terrible price to pay. Consider two linear predicates f_i, each a disjunction of n_i convex predicates with m_i linear constraints. Then $f_1 \cap \neg f_2$ consists of $n_1 \cdot n_2 \cdot m_2$ convex predicates with $m_1 + 1$ linear constraints each. Consequently, a substantial effort of an implementation must go into simplifying predicates [4].

Restriction to reachable states: If only the initial states of P and Q are of interest, the simulation relation can be initialized with the product of the reachable states of P and Q [WL97]. Often this yields a tremendous speed-up, since the state space is substantially smaller. However, if the reachable state space is too complex quite the opposite can be true. It has shown particularly advantageous to initialize R with an overapproximation of the reachable state space. That way, one profits from the speed but still retains a necessary result, not just the sufficient condition the overapproximated reachability analysis would yield by itself.

Computing the complement of the Simulation Relation: For certain cases, it can be better to iterate on the complement of the simulation relation [FV98]. This moves the difference operation from (6.1) to (6.4) and (6.10), but cannot avoid it. Since the difference operations in (6.1) turn into unions, it is possible to use overapproximation. This can also be used to force termination. R is still a simulation relation after termination, but not necessarily the largest and therefore not necessarily a witness for $P \preceq Q$. However, this is a particularly interesting option if R is used for minimization.

6.3.3 Parametric Analysis

For verifying simulation, it is the relation R between states that is parameterized. Consider the case where an automaton P has parameters w. According to (6.2), an automaton Q simulates an automaton P if

$$\underset{p}{Init_P} \cap \neg \left(\underset{q}{Init_Q}|^{p,q} \cap \underset{p,q}{R} \right) \Big\downarrow_p = \emptyset.$$

[4] The importance of simplification was also emphasized in [Ho95] because of its impact on the containment and emptiness tests in reachability analysis.

Consequently, the bad parameters, i.e., the ones violating simulation, are given by

$$B = \{w | \exists p \in Init_P : \nexists q \in Init_Q : (p,q) \in R\},$$

which can be written symbolically as

$$B = \left[\underset{p}{Init_P} \cap \neg \left(\underset{q}{Init_Q}|^{p,q} \cap \underset{p,q}{R} \right) \Big\downarrow_p \right] \Big\downarrow_w . \tag{6.12}$$

If the specification Q is parameterized, the quantification over the parameters z of Q, is existential. Then the bad parameters w in P are given by

$$B = \{w | \nexists z : \forall p \in Init_P : \exists q \in Init_Q : (p,q) \in R\},$$

and symbolically written as

$$B = \neg \left(\neg \left[\underset{p}{Init_P} \cap \neg \left(\underset{q}{Init_Q}|^{p,q} \cap \underset{p,q}{R} \right) \Big\downarrow_p \right] \Big\downarrow_{w,z} \right) \Big\downarrow_w . \tag{6.13}$$

6.4 Related Work

Compositionality and Abstraction Henrik Ejersbo Jensen et al. have worked on compositionality and abstraction in the field of timed automata and presented several case studies using the UPPAAL tool [Jen99, JLS00]. Olivero showed that a subset of linear hybrid automata[5] has a discrete abstraction that preserves ATCTL formulas, which are a real-time extension of ACTL [OSY94]. Discrete simulations for hybrid automata have also been the subject of [ADI02].

Simulation A notion of equivalence between simulating states of hybrid systems was proposed by Henzinger in [Hen96]. Pappas et al. defined simulation for linear systems with such a notion [PLS98]; later the works of Tabuada et al. generalized this concept to hybrid abstractions that preserve timed languages [TP01]. The same group also extended simulation into the realm of control systems in a general setting that takes account continuous as well as discrete interactions between hybrid systems. A general formulation was presented in [TPL01], and a formalism with explicit use of simulation relations can be found in [TPL02]. Recently, simulation relations have also been applied to discrete time systems [TP02].

Timed simulation for modal hybrid systems is presented in [WL97], where the authors also include silent, or un-observable, transitions and distinguish between weak

[5] with slopes $\neq 0$, invariants and guards of the form $\{l < x < u\}$, if the slope changes through a transition, the variable x is in the assignment, the invariant is bounded in direction opposite of the slope

and strong simulation. In *weak simulation*, a sequence of silent transitions is equivalent to a single timed transition of length zero. In strong simulation, the silent transitions must simulate the same way as other transitions, which corresponds to our notion of simulation. Silent transitions of the concrete system can easily be taken into account using Σ-simulation. Consider the inequality $P \preceq_\Sigma Q$. In Σ-simulation any transition with a label in P that is not in Q is equivalent to a silent transition. By introducing a dedicated silent label τ_P that occurs in no other automaton, this is even invariant under composition. Silent transitions in the specification Q can not be captured that way. Algorithms for computing simulation are presented in [HHK95]. Refinement of a class of timed automata with integer semantics is supported by RABBIT [BLN03].

Chapter 7

Assume-Guarantee Reasoning

Assume-guarantee reasoning is a form of compositional proof in which the specification of a subsystem is only fulfilled under assumptions about the rest of the system. For simulation-based assume-guarantee, we assume that a system P is given in components, i.e., $P = P_1 || \ldots || P_n$, and the specification can be decomposed so that there is a sub-specification for each component of the system, i.e., $Q = Q_1 || \ldots || Q_n$. The goal is to show that

$$P_1 || \ldots || P_n \preceq_{\approx} Q_1 || \ldots || Q_n. \tag{7.1}$$

The decomposition of the specification, as in Theorem 6.10, allows us to verify each sub-specification separately, i.e.,

$$P_1 || \ldots || P_n \preceq_{\approx_i} Q_i \text{ for } i = 1, \ldots, n,$$

but in the following we instead aim at exploiting the knowledge the Q_i, if fulfilled, provide about the system for reducing the complexity of the proof.

A fundamental assumption for the arguments in this chapter is that the specifications Q_i are significantly simpler than the system descriptions. Only then will the proposed methods be of advantage. The simplification can be rooted in the fact that more behavior is admitted in Q_i than in P_i. E.g., a system that expects deliveries Mondays between 10:00 and 11:00 and Wednesdays between 9:00 and 12:00 might have a simple specification in which deliveries occur daily before noon. In such cases Q_i is a conservative overapproximation of P_i, i.e., it holds that $P_i \preceq_{\approx_i} Q_i$. Compositionality, according to Theorem 6.9, then guarantees (7.1) if the equivalence relations are compatible:

$$\frac{\begin{array}{c} P_1 \preceq_{\approx_1} Q_1 \\ P_2 \preceq_{\approx_2} Q_2 \end{array}}{P_1 || P_2 \preceq_{\approx} Q_1 || Q_2}.$$

Another source of simplification can be disregard to behavior that is irrelevant with respect to the specification, or the operating conditions that it considers. E.g., a chemical

process control system usually has modes for automatic production, manual operation as well as cleaning and maintenance. The verification of a production recipe might not need to take into account what happens when the system is suddenly switched to cleaning mode. In this case, the specification might simply contain a state "cleaning", in which any behavior is admitted, so that it is still an overapproximation, or we might assume that the system simply never enters the cleaning mode. There are many other kinds of assumptions and simplifications one might like to include in order to decrease the complexity of the proofs, and in the context of hybrid systems the potential applications are still under investigation. In this chapter we deal with assumptions that do not immediately lead to conservative approximations, i.e., that for some Q_i it holds that $P_i \not\preceq_{\approx_i} Q_i$.

We adapt the assume-guarantee rules from Chapter 4 to hybrid systems by taking into account the notion of equivalence between states. For implementation purposes, we provide a symbolic computation in $\mathbb{R}^n$. The next section deals with non-circular assume-guarantee reasoning, in which a chain-rule style argument is used that is based simply on the precongruence properties. In contrast circular assume-guarantee reasoning requires the detailed inspection of the witnessing simulation relations in order to be sound, and will be covered in Sect. 7.2. A summary of related work is given in Sect. 7.3.

7.1 Non-circular Assume-Guarantee Reasoning

Non-circular assume-guarantee reasoning has the form of a chain rule, in which the knowledge about the behavior of the system in a proof step is used in the next one. Practice has shown that this form of reasoning is very natural, and corresponding concepts are widespread in engineering applications, e.g., cascade control. Recall the non-circular assume-guarantee reasoning rule from Sect. 4.1:

$$\frac{\begin{array}{c} P_1 \preceq_\Sigma Q_1 \\ Q_1 || P_2 \preceq_\Sigma Q_2 \end{array}}{P_1 || P_2 \preceq_\Sigma Q_1 || Q_2}.$$

It is valid for hybrid automata if the variable sets on both sides of the composition operator are disjoint throughout the proof, i.e., $Var_{P_i} \cap Var_{Q_j} = Var_{P_i} \cap Var_{P_j} = Var_{Q_i} \cap Var_{Q_j} = \emptyset$ for $(i,j) \in \{(1,2),(2,1)\}$. However, we must take into account equivalence, and to do so we construct a witnessing simulation relation. This leads to the following theorem:

Theorem 7.1 (Non-circular Assume-Guarantee Reasoning)**.** *Consider hybrid automata P_i, Q_i with $Var_{P_i} \cap Var_{Q_j} = Var_{P_i} \cap Var_{P_j} = Var_{Q_i} \cap Var_{Q_j} = \emptyset$ for $(i,j) \in \{(1,2),(2,1)\}$ for which holds that $P_1 \preceq_{\approx_1} Q_1$ and $Q_1 || P_2 \preceq_{\approx_1} Q_2$. Then $P_1 || P_2 \preceq_\approx Q_1 || Q_2$ follows if $((k_1,u_1),(l_1,v_1)) \in \approx_1$ and $((l_1,k_2),w),(l_2,v_2)) \in \approx_2$, $v_1 = w{\downarrow}_{Var_{Q_1}}$, implies $(((k_1,k_2),u),((l_1,l_2),v)) \in \approx$ with $u{\downarrow}_{Var_{P_1}} = u_1$, $u{\downarrow}_{Var_{P_2}} = w{\downarrow}_{Var_{P_2}}$, and $v{\downarrow}_{Var_{Q_i}} = v_i$ for $i = 1,2$.*

Proof. We follow the proof in Sect. 4.1. Let R_1 be a witnessing relation for $P_1 \preceq_{\approx_1} Q_1$, and R_2 for $Q_1||P_2 \preceq_\Sigma Q_2$. Let $p_i = (k_i, u_i)$ and $q_i = (l_i, v_i)$. With Prop. 3.11, it follows from $[\![P_1]\!] \preceq_\Sigma [\![Q_1]\!]$ that $[\![P_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_1]\!]||[\![P_2]\!]$, with a witness $R' = \{((p_1,p_2),(q_1,p_2))|(p_1,q_1) \in R_1\}$. Also, according to Prop. 3.18, $[\![P_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_1]\!]$ with a witness $R'_1 = \{((p_1,p_2),q_1)|(p_1,q_1) \in R_1\}$. Applying Prop. 6.6, this implies $[\![P_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_1||P_2]\!]$, with a witness

$$R'' = \{((p_1,(k_2,w{\downarrow}_{Var_{P_2}})),((l_1,k_2),w))|(p_1,(l_1,w{\downarrow}_{Var_{Q_1}})) \in R_1\}.$$

Using transitivity and R_2, this implies $[\![P_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_2]\!]$, with a witnessing relation $R'_2 = \{((p_1,(k_2,u_2)),q_2)|(p_1,(l_1,w{\downarrow}_{Var_{Q_1}})) \in R_1, (((l_1,k_2),w),q_2) \in R_2, u_2 = w{\downarrow}_{Var_{P_2}}\}$. By composition of the specification according to Theorem 3.20, it follows with R'_1 and R'_2 that $[\![P_1]\!]||[\![P_2]\!] \preceq_\Sigma [\![Q_1]\!]||[\![Q_2]\!]$ with a witness

$$R''' = \{((p_1,(k_2,u_2)),((l_1,v_1),q_2))|(p_1,(l_1,v_1)) \in R_1, (((l_1,k_2),w),q_2) \in R_2, \\ u_2 = w{\downarrow}_{Var_{P_2}}, v_1 = w{\downarrow}_{Var_{Q_1}}\}.$$

Applying Prop. 6.6, it finally holds that $[\![P_1||P_2]\!] \preceq_\Sigma [\![Q_1||Q_2]\!]$ with a witness

$$R = \{(((k_1,k_2),u),((l_1,l_2),v))|((k_1,u_1),(l_1,v_1)) \in R_1, (((l_1,k_2),w),q_2) \in R_2, \\ u_2 = w{\downarrow}_{Var_{P_2}}, v_1 = w{\downarrow}_{Var_{Q_1}}, u_i = u{\downarrow}_{Var_{P_i}}, v_i = v{\downarrow}_{Var_{Q_i}} \text{ for } i = 1,2\}.$$

Under the hypothesis it holds that $R \subseteq \approx$, so that R witnesses $P_1||P_2 \preceq_\approx Q_1||Q_2$. □

The following examples shall illustrate how non-circular A/G-reasoning provides a formal basis for very natural assumptions and simplifications:

Example 7.1. *Consider the tank level monitor system of Ex. 6.1 with the non-linear tank model T_s from Fig. 5.2(c). The automaton R is an abstraction of a reactor model $\hat{R}$ with non-linear dynamics, and it is guaranteed by construction that $\hat{R} \preceq R$ holds. Then $R||C \preceq Q_a||Q_c$ can be verified algorithmically, and with Theorem 7.1 and a decomposition of the specification follows $\hat{R}||C \preceq Q_a||Q_c$.*

7.2 Circular Assume-Guarantee Reasoning

The circular assume-guarantee rule from Sect. 4.2 goes as follows:

$$\frac{\begin{array}{c} P_1||Q_2 \preceq_\Sigma Q_1 \\ Q_1||P_2 \preceq_\Sigma Q_2 \\ \textit{A/G conditions} \end{array}}{P_1||P_2 \preceq_\Sigma Q_1||Q_2}.$$

If the variable sets are disjoint, i.e. if $Var_{P_1} \cap Var_{P_2} = \emptyset$ and $Var_{Q_1} \cap Var_{Q_2} = \emptyset$, we can use Prop. 6.6 to apply Theorem 4.2 to hybrid automata.

Theorem 7.2 (A/G-Σ-simulation with Equivalence)**.** *Consider hybrid automata P_i, Q_i with $Var_{P_i} \cap Var_{Q_j} = Var_{P_i} \cap Var_{P_j} = Var_{Q_i} \cap Var_{Q_j} = \emptyset$ for $(i,j) \in \{(1,2),(2,1)\}$, for which holds*

$$P_1 || Q_2 \preceq_{\approx_1} Q_1 \quad \text{and} \tag{7.2}$$
$$P_2 || Q_1 \preceq_{\approx_2} Q_2. \tag{7.3}$$

For a shorter notation, let $p_i = (k_i, u_i)$, $q_i = (l_i, v_i)$ and u, v, w_i be defined by $u{\downarrow}_{Var_{P_i}} = u_i$, $v{\downarrow}_{Var_{Q_i}} = v_i$, $w_i{\downarrow}_{Var_{P_i}} = u_i$, $w_i{\downarrow}_{Var_{Q_j}} = v_j$ for $(i,j) \in \{(1,2),(2,1)\}$. If there exist witnessing simulation relations R_1 for (7.2) and R_2 for (7.3) such that for all p_i, q_i with $(((k_i,l_j),w_i),q_i) \in R_i$ for $(i,j) \in \{(1,2),(2,1)\}$ and $\alpha \in (\Sigma_{Q_1} \cap \Sigma_{Q_2}) \cup \mathbb{R}$ there $\exists q_1' : q_1 \xrightarrow{\alpha} q_1'$ or $\exists q_2' : q_2 \xrightarrow{\alpha} q_2'$ whenever:

(i) $\alpha \in \Sigma_{P_1} \backslash \Sigma_{P_2}$ *and* $p_1 \xrightarrow{\alpha} p_1'$,

(ii) $\alpha \in \Sigma_{P_2} \backslash \Sigma_{P_1}$ *and* $p_2 \xrightarrow{\alpha} p_2'$, *or*

(iii) $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$ *and* $p_1 \xrightarrow{\alpha} p_1'$ *and* $p_2 \xrightarrow{\alpha} p_2'$, *or*

(iv) $\alpha \notin \Sigma_{P_1} \cup \Sigma_{P_2}$,

then there is a simulation relation for $P_1 || P_2 \preceq_{\approx} Q_1 || Q_2$ given by

$$R = \{(((k_1,k_2),u),((l_1,l_2),v)) | (((k_i,l_j),w_i),q_i) \in R_i \text{ for } (i,j) \in \{(1,2),(2,1)\}\}, \tag{7.4}$$

and consequently $R \subseteq \approx$ if $(((k_i,l_j),w_i),q_i) \in \approx_i$ for $(i,j) \in \{(1,2),(2,1)\}$ implies that $(((k_1,k_2),u),((l_1,l_2),v)) \in \approx$.

Proof. Because of the disjunct variables, u, v and w_i are well defined for any set of u_i and v_i. With Prop. 6.6, we obtain witnessing simulation relations R_i', which are isomorphic to R_i, for

$$[\![P_1]\!] || [\![Q_2]\!] \preceq_{\approx_1} [\![Q_1]\!] \quad \text{and}$$
$$[\![P_2]\!] || [\![Q_1]\!] \preceq_{\approx_2} [\![Q_2]\!],$$

to which we can apply Theorem 4.2. Together with the hypothesis it yields that

$$R' = \{((p_1,p_2),(q_1,q_2)) | ((p_i,q_j),q_i) \in R_i' \text{ for } (i,j) \in \{(1,2),(2,1)\}\}, \tag{7.5}$$

is a witness for $[\![P_1]\!] || [\![P_2]\!] \preceq_{\approx_1} [\![Q_1]\!] || [\![Q_2]\!]$. Applying Prop. 6.6 again we can perform the composition before the semantic TTS operator and substitute R_i for R_i' to finally obtain R. □

Since there is an infinity of states, symbolic operations must be employed similar to the computation of the simulation relation in Sect. 6.3.1. As in the computation of the simulation relation, for hybrid automata there is a distinction between discrete

and timed transitions. To simplify notation, in the following consider the continuous transition relation μ to be restricted to the invariants, i.e. for all discrete transitions:

$$\underset{v,v'}{\mu}(l,\alpha,l')\downarrow_{v} \subseteq Inv(l) \wedge \underset{v,v'}{\mu}(l,\alpha,l')\downarrow_{v'} \subseteq Inv(l') \tag{7.6}$$

This can be forced by intersecting μ with $Inv(l) \times Inv(l')$ without modifying the semantics of the automata. Checking for A/G-simulation involves the construction of simulation relations R_1 and R_2. States that violate the A/G-conditions are trimmed before or during the construction process. While conditions (i), (ii) and (iv) can be decided strictly from R_1, respectively R_2, (iii) involves both relations. Two approaches are presented: A separate trimming of R_1 and R_2 overapproximates (iii) by trimming states that have transitions in either relation, which is computationally simple, but of limited applicability in pratice. On the other hand, a composite trimming associates states p_1 and p_2 via the states q_1 and q_2 in the relations, at the price of maintaining a relation over the state space of Q_1, Q_2 and the common alphabet of all automata.

7.2.1 Separate Trimming

Discrete Transitions

Symbolically, the states in locations l_1 and l_2 of Q_1 and Q_2 that have no outgoing transitions with label a are given by

$$E_{Q_1Q_2}(l_1,l_2,a) = \neg \bigcup_{t_1=(l_1,a,\eta_1,l_1')^{v_1,v_1'}} \eta_1 \downarrow_{v_1} |^{v_1,v_2} \cap \neg \bigcup_{t_2=(l_2,a,\eta_2,l_2')^{v_2,v_2'}} \eta_2 \downarrow_{v_2} |^{v_1,v_2}. \tag{7.7}$$

The set of states that have an outgoing transition in P_1 but none in Q_1 or Q_2 is then:

$$B_1^{tr}(k,l_1,l_2,a) = \bigcup_{s=(k,a,\mu,k')^{u,u'}} \mu \downarrow_{u} |^{u,v_1,v_2} \cap E_{Q_1Q_2}(l_1,l_2,a)|^{u,v_1,v_2} \tag{7.8}$$

Timed Transitions

Since the time is always in the alphabet of the TTS of a hybrid automaton, only condition (iii) applies. The violating states are:

$$\begin{aligned} B_1^{pte}(k,l_1,l_2) = \underset{u,u',t}{\rightsquigarrow_P}(k)\downarrow_{u,t} |^{u,v_1,v_2,t} \cap \neg \underset{v_1,v_1',t}{\rightsquigarrow_{Q_1}}(l_1)\downarrow_{v_1,t} |^{u,v_1,v_2,t} \\ \cap \neg \underset{v_2,v_2',t}{\rightsquigarrow_{Q_2}}(l_2)\downarrow_{v_2,t} |^{u,v_1,v_2,t} \end{aligned} \tag{7.9}$$

$$B_1^{te}(k,l_1,l_2) = B_1^{pte}(k,l_1,l_2)\downarrow_{u,v_1,v_2} \tag{7.10}$$

Separate checking for A/G-simulation goes as follows:

1. R_1 is initialized with $R_1 := reach(P_1||Q_2||Q_1)$ or $R_1 := S_{P_1} \times S_{Q_2} \times S_{Q_1}$.
2. For all (k, l_2, l_1), $a \in \Sigma_{P_1} \cap \Sigma_{Q_1} \cap \Sigma_{Q_2}$:
   $R_1(k, l_2, l_1) := R_1(k, l_2, l_1) \cap \neg B_1^{tr}(k, l_1, l_2, a) \cap \neg B_1^{te}(k, l_1, l_2)$, which satisfies conditions (i) and (iii).
3. For all (k, l_2, l_1), $a \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \backslash (\Sigma_{P_1} \cup \Sigma_{P_2})$:
   $R_1(k, l_2, l_1) := R_1(k, l_2, l_1) \cap \neg E_{Q_1 Q_2}(l_1, l_2, a)$, which satisfies condition (iv).
4. $R_1 := GetSimRel_{P_1||Q_2, Q_1}(R_1)$.
5. Repeat 1) - 4) for R_2 analogously.
6. If containment of the initial states in R is guaranteed, then $P_1||P_2 \preceq Q_1||Q_2$.

7.2.2 Composite Trimming

Discrete Transitions

The sets of critical labels and states in R_1 and R_2 that could violate the A/G-conditions are projected onto the state space of Q_1 and Q_2:

$$D_2^{tr}(l_1, l_2, a) = \bigcup_k \left(\underset{v_1, u, v_2}{R_2(l_1, k, l_2)}|^{u, v_1, v_2} \cap \underset{u, v_1, v_2}{B_2^{tr}}(k, l_1, l_2, a) \right) \Big\downarrow_{v_1, v_2}, \tag{7.11}$$

analogously for D_1. The potentially violating states of condition (iii) in R_1 are then:

$$B_1^{trc}(k, l_1, l_2, a) = \bigcup_{s=(k, a, \mu, k')^{u, u'}} \mu \downarrow_u |^{u, v_1, v_2} \cap D_2^{tr}(l_1, l_2, a)|^{u, v_1, v_2}. \tag{7.12}$$

Timed Transitions

$$D_2^{te}(l_1, l_2) = \bigcup_k \left(\underset{v_1, u, v_2}{R_2(l_1, k, l_2)}|^{u, v_1, v_2, t} \cap \underset{u, v_1, v_2, t}{B_2^{pte}}(k, l_1, l_2) \right) \Big\downarrow_{v_1, v_2, t}, \tag{7.13}$$

$$B_1^{tec}(k, l_1, l_2) = \left(\underset{u, u', t}{\rightsquigarrow_P}(k) \downarrow_{u,t} |^{u, v_1, v_2, t} \cap D_2^{te}(l_1, l_2)|^{u, v_1, v_2, t} \right) \Big\downarrow_{u, v_1, v_2} \tag{7.14}$$

Composite checking for A/G-simulation goes as follows:

1. R_1 is initialized with $R_1 := reach(P_1||Q_2||Q_1)$ or $R_1 := S_{P_1} \times S_{Q_2} \times S_{Q_1}$.
2. For all (k, l_2, l_1), $a \in \Sigma_{P_1} \cap \Sigma_{Q_1} \cap \Sigma_{Q_2} \backslash \Sigma_{P_2}$:
   $R_1(k, l_2, l_1) := R_1(k, l_2, l_1) \cap \neg B^{tr1}(k, l_1, l_2, a)$, which satisfies condition (i).

3. For all (k, l_2, l_1), $a \in \Sigma_{Q_1} \cap \Sigma_{Q_2} \setminus (\Sigma_{P_1} \cup \Sigma_{P_2})$:
   $R_1(k, l_2, l_1) := R_1(k, l_2, l_1) \cap \neg E_{Q_1 Q_2}(l_1, l_2, a)$, which satisfies condition (iv).
4. $R_1 := GetSimRel_{P_1||Q_2, Q_1}(R_1)$.
5. Repeat 1)-4) for R_2 analogously.
6. Compute $D_1^{tr}(l_1, l_2, a)$, $D_1^{te}(l_1, l_2)$, $D_2^{tr}(l_1, l_2, a)$, $D_2^{te}(l_1, l_2)$.
   If $D_1^{tr}(l_1, l_2, a) \cap D_2^{tr}(l_1, l_2, a) = \emptyset$ and $D_1^{te}(l_1, l_2) \cap D_2^{te}(l_1, l_2) = \emptyset$ goto 10).
7. For all (k, l_2, l_1), $a \in \Sigma_{P_1} \cap \Sigma_{P_2} \cap \Sigma_{Q_1} \cap \Sigma_{Q_2}$:
   $R_1(k, l_2, l_1) := R_1(k, l_2, l_1) \cap \neg B_1^{trc}(k, l_1, l_2, a) \cap \neg B_1^{tec}(k, l_1, l_2)$, which satisfies condition (iii).
8. $R_1 := GetSimRel_{P_1||Q_2, Q_1}(R_1)$.
9. Repeat 7) - 8) for R_2 analogously.
10. If containment of the initial states in R is guaranteed, then $P_1||P_2 \preceq Q_1||Q_2$.

7.2.3 Checking the Initial States

To finalize the A/G-proof, it must be shown that all the initial states of $P_1||P_2$ have a matching initial state of $Q_1||Q_2$ in R, i.e., that for all $(p_1, p_2) \in Init_{P_1} \times Init_{P_2}$ there exist $(q_1, q_2) \in Init_{Q_1} \times Init_{Q_2}$ such that $(p_i, q_1, q_2) \in R_i$ for $i = 1, 2$. It must hold for all locations $(k_1, k_2) \in Init_{P_1} \times Init_{P_2}$ that

$$\begin{aligned}
&\underset{u_1}{Init_{P_1}}(k_1)|^{u_1,u_2} \cup \underset{u_2}{Init_{P_1}}(k_2)|^{u_1,u_2} \cap \\
&\neg \bigcup_{l_1,l_2} \Bigg(\underset{v_1}{Init_{Q_1}}(l_1)|^{u_1,u_2,v_1,v_2} \cap \underset{v_2}{Init_{Q_1}}(l_2)|^{u_1,u_2,v_1,v_2} \\
&\qquad \cap \underset{u_1,v_2,v_1}{R_1}(k_1, l_2, l_1)|^{u_1,u_2,v_1,v_2} \cap \underset{u_2,v_1,v_2}{R_2}(k_2, l_1, l_2)|^{u_1,u_2,v_1,v_2} \Bigg) \Bigg\downarrow_{u_1,u_2} = \emptyset. \quad (7.15)
\end{aligned}$$

7.3 Related Work

Tasiran et al. applied A/G-reasoning to a class of timed systems with synchronous composition and use non-blocking as a requirement to break the circularity [TAKB96]. A non-circular approach to A/G-reasoning has been applied by Hooman to distributed real-time systems using the theorem prover PVS [Hoo98]. Xu and Swarup use a duration calculus to verify timed systems, and also provide a brief survey of compositional rules in [XS98]. Furia uses TRIO, a typed linear metric logic, and PVS in his thesis to compositionally verify timed systems and applies it to a reservoir-controller system [Fur03].

The compositional verification of hybrid systems is still a developing field. Alur and Henzinger extend in [AH97] the A/G-principle from reactive systems to timed and hybrid systems. They check for receptiveness in order to break circularity and show that the check is complete for EXPTIME. Thomas Henzinger et al. applied the assume-guarantee principle to hybrid systems in [HQR98] and have a developed hierarchical model that supports nesting of parallel and serial composition [HMP01]. They use the language MASACCIO to verify two cooperating robots. Rajeev Alur et al. use the modeling language CHARON for the modular design of interacting hybrid systems addressing different aspects of hierarchy [ADE$^+$01]. In [Hoo93], Hooman uses modified Hoare-Triples to compositionally verify a water level monitoring system. Together with Vitt he applied a similar approach to a steam boiler control system using PVS [VH96].

Part III

Hybrid Systems with Continuous Interaction

Introduction

In the previous part we showed that hybrid automata can be analyzed compositionally with labeled transition system semantics as long as they share no variables. In this part we will show weaker compositional rules when variables are shared, which entails a drastic overapproximation of their interaction. We identify a class of automata in which the shared variables are unrestricted in such a way that this overapproximation has no effect. This class is of practical relevance, since it can model digitally controlled plants.

The core of applying compositional methods to hybrid systems with shared variables lies in two restrictions: using an I/O-framework to control access to variables and identify which variables are equivalent between the system and the specification, and using timed transition system (TTS) semantics to obtain a computable problem. In our I/O-framework, each variable is "owned" as a control variable by exactly one automaton. In all other automata, it is considered an input variable that can change arbitrarily at any time. Most importantly, a transition in which the owner does not participate cannot change the its value. In certain cases this allows us to conclude that a transition that is possible in an automaton will also be possible in the composition with another, which is a key to compositionality. As a semantic basis we use timed transition systems (TTS), which abstract from the continuous change of variables by existential quantification. A change in a variable over a certain time is possible in the TTS if there exists some activity in the hybrid automaton for such a change in that time. This allows us to obtain a finitary representation of many systems, e.g., using polyhedra. Applying TTS-semantics leads to a complete overapproximation of the continuous interaction that can prevent the application of compositional methods in many cases, but we identify two important classes in which this is not the case: for hybrid automata with unrestricted inputs, and for linear hybrid automata with convex invariants TTS-simulation is compositional. The adaptation of the assume-guarantee rule of the previous parts leads to the fundamental result additional conditions, the A/G-conditions, must ensure that the specification allows transitions on common labels and time elapse, and that transitions on independent labels do not violate the invariants of the specification. We provide A/G-conditions that can be checked in one pass on the simulation relations, and that can be checked with a complexity that is insignificant compared to the complexity of checking simulation for the individual components.

In the next chapter, we will introduce our hybrid I/O-automaton model, and pro-

pose run semantics as well as hybridized TTS semantics that retain the location/variable separation. They differ from the semantics of the previous parts with a special environment label ε that serves to "encode" the invariants of hybrid automata using labels. If one automaton simulates another, it will also have equivalent, or more relaxed, invariants. The environment action holds a special place in the parallel composition operator that prevents independent transitions of one automaton to violate the invariant of the other. We also show consistency between run and TTS semantics. In Chapter 9, we define simulation based on the TTS semantics, and relate it to the trace based simulation used in literature – the main difference being that trace based simulation is not computable algorithmically in the general case. We show that TTS-simulation is consistent with trace simulation since it is implied by it. Because of the shared variables, compositionality as in the previous parts does not hold for TTS-simulation. We propose a weak compositional rule that requires as an additional condition that the composition of the timed transition systems of the specification is equivalent to the composition of the hybrid automata models. We call such systems for which this holds *TTS-compositional*. This is generally not the case, but we identify two important classes of systems that are TTS-compositional: systems with unrestricted inputs, and linear hybrid automata with convex invariants.

The overapproximation of TTS-semantics, although irrelevant for TTS-compositional systems, is also prevalent in our adaption of assume-guarantee reasoning to HIOA, presented in Chapter 10. The assume/guarantee rule is similar in structure to the ones in the previous parts of this thesis, but adds a condition for the environment action that amounts to checking that the invariants agree with the specification.

We conclude this part with an overview on PHAVer in Chapter 11. PHAVer is a verification tool for linear hybrid automata that differs from other existing tools through exact and robust arithmetic, and an experimental implementation of simulation checking that allows us to perform compositional and assume/guarantee reasoning. PHAVer also has the capability to conservatively approximate affine dynamics, i.e., of the form $\dot{x} = Ax + b$, on the fly. The computational complexity is managed by limiting the number of bits and the number of constraints in polyhedra. The number of polyhedra is controlled by partitioning the state space into cells, and applying the convex hull to the states in a cell. Experimental results for a navigation benchmark and a tunnel diode circuit demonstrate the success of the approach. While in the given timeframe we were not able to apply these methods to simulation checking, we believe they will be vital to the success of future implementations. Our conclusions for Part III can be found in Sect. 12.3, pp. 169.

Chapter 8

Modeling with Hybrid I/O-Automata

The existing theory of hybrid I/O-automata as developed by Lynch, Segala and Vaandrager [LSV03] is very powerful, but it would be cumbersome to apply our framework from the previous parts in that formalism. Since our focus is on obtaining a simple, computable framework for compositional reasoning, we propose a simpler concept of hybrid I/O-automata that, while not as powerful as the one of Lynch et al., allows us to reuse many results of the previous parts of this thesis. We do not impose an I/O-structure on the synchronization labels in order to avoid overhead that is irrelevant to the central ideas of this thesis. The directed communication associated with an I/O-structure on the labels can be modeled in our framework as outlined in Sect. 2.3, so that the generality of our approach is not restricted by omitting such a structure.

In the next section, we introduce our model of hybrid I/O-automata, which is a simple extension of the automata used in Part II and not as powerful as, e.g., the HIOA of Lynch et al. This allows us to use results from the previous parts and keep the proofs simple. In Sect. 8.2 we present the timed transition system semantics. Since we consider equivalence between the automata based on the valuation of variables, we define hybrid labeled transition systems (HLTS) that retain the location/valuation structure of hybrid automata. The semantics of a HIOA are then given as a timed transition system (TTS), which attributes a HLTS to each HIOA. The composition of the TTS is different from the composition of the HIOA, which we illustrate with some examples. Their relationship will be formalized in the next chapter with the help of simulation relations.

8.1 Hybrid I/O-Automata

We extend the hybrid automata model from Part II by differentiating between state, input and output variables. The output variables are a subset of the control variables, and can be declared as input in other automata. The controlled variables of an automaton cannot be changed by another automaton, which will reflect in the parallel composition of automata. We proceed with our definition of hybrid I/O-automata and their

trace semantics. The relation to other existing work will be addressed in Sect. 8.3.

Definition 8.1 (Hybrid I/O-Automaton)**.** *A* **hybrid input/output-automaton** *(HIOA)* $H = (Loc, Var_C, Var_I, Var_O, Lab, \rightarrow, Act, Inv, Init)$ *consists of the following:*

- *A finite set Loc of locations.*
- *Finite sets of* **controlled** *variables* Var_C*,* **input** *variables* Var_I*,* $Var_C \cap Var_I = \emptyset$*, and* **output** *variables* $Var_O \subseteq Var_C$*. Let* $Var = Var_C \cup Var_I$ *and let the* **external** *variables be* $Var_E = Var_I \cup Var_O$*. A pair* (l, v) *of a location and a valuation is a* **state** *of the automaton and the* **state space** *is* $S_H = Loc \times V(Var)$*.*
- *A finite set Lab of synchronization labels.*
- *A finite set of discrete transitions* $\rightarrow \subseteq Loc \times Lab \times 2^{V(Var)\times V(Var)} \times Loc$*. A transition* $(l, a, \mu, l') \in \rightarrow$ *is also written as* $l \xrightarrow{a,\mu}_H l'$*.*
- *A mapping* $Act : Loc \rightarrow 2^{Ats(Var)}$ *from locations to activities.*
- *A mapping* $Inv : Loc \rightarrow 2^{V(Var)}$ *from locations to sets of valuations.*
- *A set* $Init \subseteq Loc \times V(Var)$ *of initial states such that* $(l, v) \in Init \Rightarrow v \in Inv(l)$*.*

As before, the semantics of a hybrid I/O-automaton is captured by the concept of a *run*, which is any admissible sequence of changes in the state of the automata, and of an *execution*, which is a run that starts in of the initial states. We consider the inputs for the automaton open, i.e., possibly changing spontaneously and arbitrarily within the invariant, and account for them with environment transitions that have a dedicated label ε.[1]

Definition 8.2 (Run, Execution)**.** *A* **run** σ *is a finite or infinite sequence of states* (l_i, v_i) *and labels* $\alpha_i \in Lab \cup \varepsilon \cup \left(\mathbb{R}^{\geq 0} \times Ats(Var)\right)$*,*

$$\sigma = (l_0, v_0) \xrightarrow{\alpha_0} (l_1, v_1) \xrightarrow{\alpha_1} (l_2, v_2) \xrightarrow{\alpha_1} \ldots, \tag{8.1}$$

satisfying that for all $i \geq 0$ *holds* $v_i \in Inv(l_i)$ *and either*

- $\alpha_i \in Lab$ *and there is a transition* $l_i \xrightarrow{\alpha_i,\mu}_H l_{i+1}$ *with* $(v_i, v_{i+1}) \in \mu$*, or*
- $\alpha_i = \varepsilon$ *and* $l_i = l_{i+1}$*,* $v_i{\downarrow}_{Var_C} = v_{i+1}{\downarrow}_{Var_C}$*, or*
- $\alpha_i =: (t_i, f_i)$ *and* $f_i \in Act(l_i)$*,* $l_i = l_{i+1}$*,* $f_i(0) = v_i$*,* $f_i(t_i) = v_{i+1}$ *and for all* t'*,* $0 \leq t' \leq t_i$*, holds* $f_i(t') \in Inv(l_i)$*.*

[1] These are similar to the stutter transitions in [ACH+95], where they are included in the syntactic definition of automata. We feel they should be part of the semantics, but see no tangible difference.

An **execution** *is a run σ that starts in one of the initial states, i.e., $(l_0, v_0) \in Init$. A state (l, v) is called* **reachable** *if there exists an execution with $(l, v) = (l_i, v_i)$ for some $i \geq 0$.*

In the I/O-setting we can differentiate between behavior of an automaton that is visible to the outside world, i.e., inputs and outputs, and internal behavior that is considered invisible to the outside world. Since the focus of our investigation is the continuous interaction, we have no such distinction for synchronization labels. In general, we are interested in the output a system produces for a given input, and also decide equivalence between systems based on the outputs they produce for the same input. We therefore consider sequences of inputs and outputs together as the externally visible behavior of the system. Formally, the continuous evolution of the in- and output variables and the sequence of labeled transitions defines a *trace*:

Definition 8.3 (Trace)**.** *A* **trace** *$\tau(\sigma)$ of a run $\sigma = (l_0, v_0) \xrightarrow{\alpha_0} (l_1, v_1) \xrightarrow{\alpha_1} (l_2, v_2) \xrightarrow{\alpha_1} \ldots$ is the sequence of labels $\beta_i \in Lab \cup \varepsilon \cup \left(\mathbb{R}^{\geq 0} \times Ats(Var_E)\right)$ defined by*

- *$\beta_i = \alpha_i$ when $\alpha_i \in Lab \cup \varepsilon$, and*
- *$\beta_i = (t_i, f_i{\downarrow}_{Var_E})$ when $\alpha_i \in \mathbb{R}^{\geq 0} \times Ats(Var)$, where $\alpha_i =: (t_i, f_i)$.*

We say that τ is a trace of P if it is a trace of any execution of P and denote the set of traces of P by $traces(P)$.

The projection operator extends to traces in the straightforward way: $\tau' = \tau{\downarrow}_{Var}$is the trace with the same labels a_i, the same times t_i and $g'_i = g_i{\downarrow}_{Var}$, where i ranges over the length of the trace.

Hybrid automata are combined by a parallel composition operator, which enables the modular modeling of complex systems. For I/O-automata, a notion of compatibility is needed. In the following, let $H_i = (Loc_i, Var_{Ci}, Var_{Ii}, Var_{Oi}, Lab_i, \rightarrow_i, Act_i, Inv_i, Init_i)$, $i = 1, 2$, be hybrid I/O-automata.

Definition 8.4 (Compatibility)**.** *Hybrid I/O-automata H_1, H_2 are* **compatible** *if their control variables, Var_{C1} and Var_{C2}, are disjoint, and any inputs from each other's variables are part of the output variables, i.e., $Var_{C1} \cap Var_{C2} = \emptyset$ and $Var_{Ii} \cap Var_{Cj} \subseteq Var_{Oj}$ for $(i, j) \in \{(1, 2), (2, 1)\}$.*

The parallel composition operator determines how two automata interact. Changes in the continuous variables must be matched in both, and a discrete transition can only change a variable if the automaton, who has it as a control variable, agrees.

Definition 8.5 (Parallel Composition of HIOA)**.** *Given compatible HIOA H_1, H_2, their* **parallel composition** *$H_1 \,||\, H_2$ is the HIOA H with*

- *$Loc = Loc_1 \times Loc_2$,*

- $Var_C = Var_{C1} \cup Var_{C2}$, $Var_O = Var_{O1} \cup Var_{O2}$, $Var_I = (Var_{I1} \cup Var_{I2}) \setminus Var_O$,
- $Lab = Lab_1 \cup Lab_2$,
- $f \in Act(l_1, l_2)$ *iff* $f{\downarrow}_{Var_i} \in Act_i(l_i)$, $i = 1,2$,
- $v \in Inv(l_1, l_2)$ *iff* $v{\downarrow}_{Var_i} \in Inv_i(l_i)$, $i = 1,2$, *and*
- $(l_1, l_2) \xrightarrow{a,\mu}_H (l_1', l_2')$ *with* $\mu = \{(v,v') | (v{\downarrow}_{Var_i}, v'{\downarrow}_{Var_i}) \in \mu_i, i = 1,2\}$ *iff for* $i = 1,2$*:*
  - $a \in Lab_i$ *and* $l_i \xrightarrow{a,\mu_i}_i l_i'$, *or*
  - $a \notin Lab_i$ *and* $l_i = l_i'$, $\mu_i = \{(v,v') | v{\downarrow}_{Var_{Ci}} = v'{\downarrow}_{Var_{Ci}}\}$,
- $((l_1, l_2), v) \in Init$ *iff* $(l_i, v{\downarrow}_{Var_i}) \in Init_i$, $i = 1,2$.

8.2 Hybrid Labeled Transition Systems

We use *hybrid labeled transition systems* as a formal model for *timed transition systems*, which will provide the semantic basis for hybrid automata. They preserve most of the structure of the hybrid automaton while abstracting from the continuous activities and invariants. In contrast to the LTS of Part II, the retain a location/valuation structure that will simplify many proofs and allow us to define equivalence directly based on valuations. The separation of input- and output variables is preserved to obtain consistent semantics. For shared variables, we need to take into account that the input variables can change arbitrarily at any time. Therefore we introduce ε-transitions that will represent such a change in the semantics of HIOA. [2] These transitions will be of vital importance in the proofs on compositionality and circular assume-guarantee reasoning.

Definition 8.6 (Hybrid Labeled Transition System)**.** *A* **hybrid labeled transition system** $L = (Loc, Var_C, Var_I, Var_O, \Sigma, \rightarrow_L, Init)$ *consists of*

- *a finite set Loc of locations,*
- *finite disjoint sets* Var_I, Var_C *of input and control variables, and a set* $Var_O \subseteq Var_C$ *of output variables; let* $Var = Var_I \cup Var_C$, $Var_E = Var_I \cup Var_O$,
- *a set* Σ *of labels that contains a special label* ε *called* **environment label**,
- *a transition relation* $\rightarrow \subseteq Loc \times V(Var) \times \Sigma \times V(Var) \times Loc$, *and*
- *a set of initial states* $Init \subseteq Loc \times V(Var)$.

[2] The environment label is closely related to the stutter label in [ACH+95], see Sect. 8.3.

Similarly to hybrid automata, HLTSs interact by synchronizing on common labels, and control variables can only change if the automaton that owns them participates in the transition. However, contrary to the parallel composition used in for LTS in Parts I and II, transitions with non-common labels are no longer independent. They must synchronize with environment transitions, which will be used in the semantic definition of HIOA to preserve the invariants through simulation.

Definition 8.7 (Parallel Composition of HLTS)**.** [3] *Given HLTSs* $L_i = (Loc_i, Var_{Ci}, Var_{Ii}, Var_{Oi}, \Sigma_i, \rightarrow_i, Init_i)$, $i = 1,2$, *with disjoint sets of control variables, their* **parallel composition** $L_1 \,||\, L_2$ *is the HLTS* $L = (Loc, Var_C, Var_I, Var_O, \Sigma, \rightarrow_L, Init)$ *with*

- $Loc = Loc_1 \times Loc_2$,
- $Var_C = Var_{C1} \cup Var_{C2}$, $Var_O = Var_{O1} \cup Var_{O2}$, $Var_I = (Var_{I1} \cup Var_{I2}) \setminus Var_O$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $((l_1, l_2), v) \xrightarrow{\alpha}_L ((l_1', l_2'), v')$ *iff for* $i = 1$ *and* $i = 2$*:*
  - $\alpha \in \Sigma_i$ *and* $(l_i, v{\downarrow}_{Var_i}) \xrightarrow{\alpha}_{L_i} (l_i', v'{\downarrow}_{Var_i})$ *or*
  - $\alpha \notin \Sigma_i$ *and* $(l_i, v{\downarrow}_{Var_i}) \xrightarrow{\varepsilon}_{L_i} (l_i', v'{\downarrow}_{Var_i})$, *and*
- $Init = \{((l_1, l_2), v) | (l_i, v{\downarrow}_{Var_i}) \in Init_i, i = 1,2\}$.

The behavior of a hybrid automaton is associated with a hybrid labeled transition system, its *timed transition system* (TTS). Such TTSs were already introduced in Sect. 5.3 to define the semantics of the hybrid automata of Part II, whose focus was on compositional reasoning with disjunct variables.

Definition 8.8 (Timed Transition System)**.** *The* **timed transition system** *of a HIOA* H *is the HLTS* $[\![H]\!] = (Loc, Var_C, Var_I, Var_O, \Sigma, \rightarrow_{[\![H]\!]}, Init)$, *where* $\Sigma = Lab \cup \mathbb{R}^{\geq 0} \cup \varepsilon$ *and*

- $(l, v) \xrightarrow{a}_{[\![H]\!]} (l', v')$ *iff* $l \xrightarrow{a,\mu}_H l', (v, v') \in \mu, v \in Inv(l), v' \in Inv(l')$ *(discrete transitions),*
- $(l, v) \xrightarrow{t}_{[\![H]\!]} (l', v')$ *iff* $l = l'$ *and there exists* $f \in Act(l), f(0) = v, f(t) = v'$, *and* $\forall t', 0 \leq t' \leq t : f(t') \in Inv(l)$ *(timed transitions),*
- $(l, v) \xrightarrow{\varepsilon}_{[\![H]\!]} (l', v')$ *iff* $l = l'$, $v{\downarrow}_{Var_C} = v'{\downarrow}_{Var_C}, v, v' \in Inv(l)$ *(environment transitions).*

[3] This definition of parallel composition is similar to that in [Hen96], where independent transitions must synchronize with transitions of zero time elapse instead of the environment transtitions we use.

The loss of information about the activities of the hybrid automaton can lead to a different behavior when comparing the composition of timed transition systems with the timed transition system of the composed hybrid automata. This will be discussed in more detail in Sect. 9, when the comparison of systems is formalized by the notion of simulation relations.

We now show that the timed transition system semantics are consistent with our run-semantics. The proof is straightforward, since the existence of a run segment immediately corresponds to the existence of a transition in the TTS, and vice versa, because their definitions are practically identical.

Proposition 8.9. *In any HIOA H there is a run $\sigma = p_0 \xrightarrow{\alpha_0} p_1 \xrightarrow{\alpha_1} p_1 \cdots$ if and only if there is a run $\sigma' = p_0 \xrightarrow{\beta_0}_{[\![H]\!]} p_1 \xrightarrow{\beta_1}_{[\![H]\!]} p_2 \cdots$ in $[\![H]\!]$, where $\beta_i =: t_i \in \mathbb{R}^{\geq 0}$ iff $\alpha_i =: (t_i, f_i) \in \mathbb{R}^{\geq 0} \times Ats(Var)$, and $\alpha_i = \beta_i$ otherwise.*

Proof. Both directions follow immediately from the definitions of runs and the timed transition system. Any transition in a run of H also fulfills the definition of a transitions in $[\![H]\!]$. Conversely, any transition in $[\![H]\!]$ implies either a discrete or environment transition in H, or the existence of an activity f_i such that a timed transition exists in H. □

Corollary 8.10. *A state p is reachable in H if and only if it is reachable in $[\![H]\!]$.*

We end this section with examples that shall illustrate the difference between the composition of two HIOA and the composition of their TTS. Intuitively, a HLTS only carries information about which states can transition to which, and whether by time elapse or by a discrete transition, but not about the trajectories taken to get there. Consequently, the composition of two timed transition systems of hybrid automata with shared variables is an overapproximation of the hybrid system's behavior because non-matching trajectories in the systems can be paired. As demonstrated by the following example, a continuous input is completely abstracted away:

Example 8.1. *Consider hybrid automata P_1 and P_2, both with a single location k, respectively l, no labels or discrete transitions. Let P_1 have a state and output variable u, and activities defined implicitly by $\dot{u} = au$, a constant $a < 0$, no invariant, and an initial state (l, u_0). Let P_2 have a control variable x and activities $\dot{x} = u$, i.e., unrestricted activities for u, an invariant $u \geq c$ with a constant c and an initial state (l, x_0). Then $[\![P_1]\!]$ has timed transitions $(k, u) \xrightarrow{t}_{[\![P_1]\!]} (k, e^{at}u)$, and those of $[\![P_2]\!]$ are given by*

$$\exists f(t) \in Ats(u), f(t) \geq c : (l, (x, u)) \xrightarrow{t}_{[\![P_2]\!]} (l, (x + \int_0^t f(\tau) d\tau, f(t))),$$

which corresponds to

$$(l, (x, u)) \xrightarrow{t}_{[\![P_2]\!]} (l, (x', u')) \text{ for any } t > 0, (x', u') \in \mathbb{R}^2, x' \geq x + ct, u' \geq c.$$

The timed transitions of $[\![P_1||P_2]\!]$ *are*

$$((k,l),(x,u)) \xrightarrow{t}_{[\![P_1||P_2]\!]} ((k,l),(x+(e^{at}-1)u,e^{at}u)),$$

with $(e^{at}-1)u \geq ct$ *and* $e^{at}u \geq c$*. The timed transitions of* $[\![P_1]\!]|||[\![P_2]\!]$ *are*

$$((k,l),(x,u)) \xrightarrow{t}_{[\![P_1]\!]|||[\![P_2]\!]} ((k,l),(x',e^{at}u))$$

for any $t > 0, x' \in \mathbb{R}, x' \geq x+ct, e^{at}u \geq c$*. The value of x became dissociated from the value of u in the target states. While the latter is certainly a gross overapproximation, we can still deduce some properties of* $P_1||P_2$ *from looking at* $[\![P_1]\!]|||[\![P_2]\!]$*, e.g. a check for simulation with a parameterized specification could yield that* $x \geq x_0+ct$*,* $c \leq u < u_0$ *for all* $t > 0$*.*

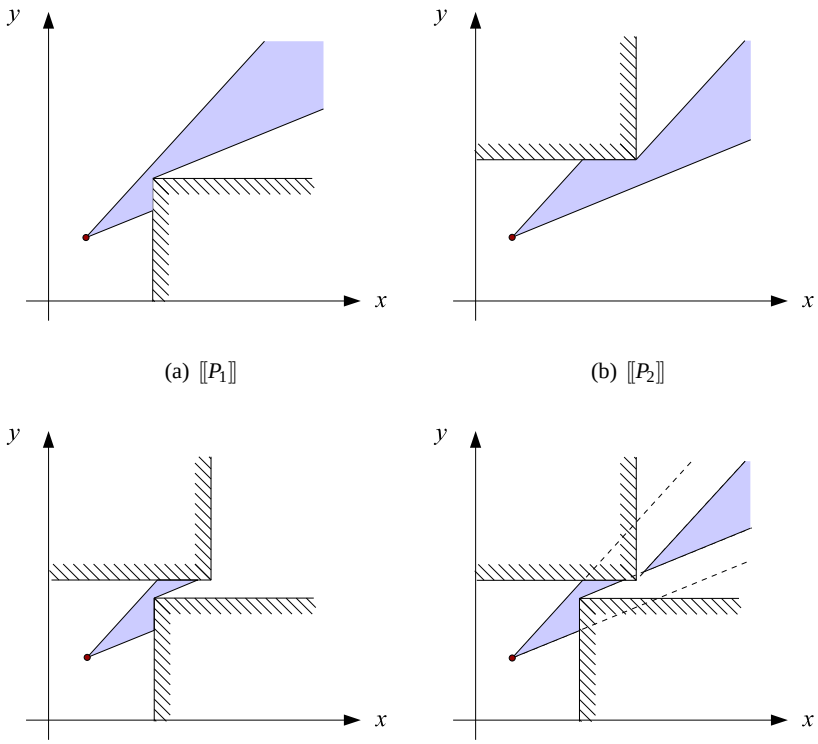
Another example shall illustrate how the composition of timed transition systems does not respect a non-convex invariant of the composed hybrid systems:

Example 8.2. *Consider automata* P_1 *and* P_2 *with invariants shown as hatched regions in Figs. 8.1(a) and 8.1(b). The shaded regions show which states are in the timed transition relation with the initial state, represented by a dot. While the TTS* $[\![P_1]\!]$ *and* $[\![P_2]\!]$ *respect the invariants of* P_1*, respectively* P_2*, their composition Fig. 8.1(d) shows a spurious set of states that does not exist in the TTS of* $P_1||P_2$*, shown in Fig. 8.1(c).*

8.3 Related Work

The theory of hybrid I/O-automata has been developed extensively by Lynch, Segala and Vaandrager [LSVW96, LSV01, LSV03]. Their framework is very powerful and general, and based on practically arbitrary trajectories of a set of variables, which can have different dynamic types. Since our focus is on obtaining a simple, computable framework for compositional reasoning, we propose a simple concept of I/O-automata that, while not as powerful as the one of Lynch et al., allows us to reuse many results of the previous parts of this thesis, and admits simple proofs. It is an extension of the hybrid automata of Part II in the direction of [LSV03]. More precisely, it is related to the pre-HIOA in [LSV03], since we do not impose enabling of input actions or input trajectories. It differs in that we allow the set of initial states to be empty, do not impose axioms on the set of trajectories and do not impose restrictions on the discrete transitions. Moreover, we do not differentiate between internal and external actions.

Since the basic components of our hybrid automaton model was already discussed in Sect. 5.4, we only discuss the extensions. In our model the output variables are part of the controlled variables, while in [LSV03] they are part of the external variables. The hybrid automata in [ACH+95] also have controlled variables, but have no distinguished output variables. The parallel composition of hybrid automata in [ACH+95]



(a) $[\![P_1]\!]$ (b) $[\![P_2]\!]$

(c) $[\![P_1||P2]\!]$ (d) $[\![P_1]\!]||[\![P_2]\!]$

Figure 8.1: Composition of timed transition systems with non-convex invariant

is semantically but not syntactically identical. It achieves the effect of constant controlled variables in independent transitions using stutter transitions.[4] The stutter label τ in [ACH+95] is part of their syntactic definition of hybrid automata. We have moved this label to the semantic of hybrid automata, i.e., the definition of runs – through which it finds its way into trace-simulation – and hybrid labeled transition systems (HLTSs), along with a corresponding definition of parallel composition for HLTSs. There is essentially no difference, and one could rewrite this part of this thesis with the syntax of hybrid automata of [ACH+95]. Note that the parallel composition of HLTS would still have to keep its present form. Also note that the environment label ε is different from a silent label, which are considered invisible with respect to simulation – then called weak simulation. For compositionality to hold, one must use strict simulation with the environment label, while it might still desirable to have in addition a silent label with respect to which weak simulation is applied. Our parallel composition operator for HLTS is similar to that for LTS in [Hen96]. There, an independent transition must synchronize with a transition of zero time elapse, instead of the environment transitions that we use.

[4] A stutter transition in [ACH+95] is a self-loop with a dedicated label τ for which the controlled variables remain constant while the others can change arbitrarily.

Chapter 9

Simulation Relations for Hybrid I/O-Automata

A simulation relation between automata P and Q relates states in P to states of Q that show the same, or more, behavior. The automaton Q is considered to contain the behavior of P, in the sense of an overapproximation, if every initial state of P finds a corresponding initial state of Q in the relation. This is denoted as $P \preceq Q$. We will introduce two definitions of simulation: simulation that is based on traces, and simulation based on the timed transition system (TTS) semantics. Trace based simulation is compositional but has no finitary representation, which prohibits its algorithmical implementation. Simulation based on the TTS semantics on the other hand is compositional only in special cases. We identify two important classes: hybrid automata with unrestricted inputs, and linear hybrid automata with convex invariants.

In the following section, we give definitions for trace- and TTS-simulation, and discuss their relation. In Sect. 9.2, we give a weak compositionality rule for TTS-simulation, and identify classes of hybrid systems for which TTS-simulation is compositional. The chapter concludes with Sect. 9.3 on related work.

9.1 Trace- and TTS-Simulation

In Part II, we determined which variables in a system should correspond to which variables in a specification by an equivalence relation. Compositionality was only given if the equivalence relations agreed on the relevant states. For HIOA, we consider input- and output variables equivalent, and to obtain compositionality we impose that a system and its abstraction must have the same output variables, and the abstraction can not have more inputs than the system. Also, their alphabets must range over the same labels. To accommodate differing alphabets, we could use Σ-simulation as in the previous parts. We do not do so for the sake of clarity, as this would unnecessarily complicate the formalism. If the above conditions are fulfilled, we call the automaton *comparable* to its abstraction. In the following, let P and Q be HLTS $P = (Loc_P, Var_{CP},$

Var_{IP}, Var_{OP}, Σ_P, $\rightarrow_P$, $Init_P$) and $Q = (Loc_Q, Var_{CQ}, Var_{IQ}, Var_{OQ}, \Sigma_Q, \rightarrow_Q, Init_Q)$. We define formally:

Definition 9.1 (Comparability)**.** *For HLTS P,Q, P is* **comparable** *with Q if* $\Sigma_P = \Sigma_Q$, $Var_{IQ} \subseteq Var_{IP}$ *and* $Var_{OQ} = Var_{OP}$. *For HIOA* H_1,H_2, H_1 *is comparable with* H_2 *if* $[\![H_1]\!]$ *is comparable with* $[\![H_2]\!]$.

The externally observable behavior of a hybrid I/O-automaton is defined by its traces. We define simulation based on traces and will later compare it to a definition based on TTS-semantics. The following definition is an modification of the one given in [LSV03], see Sect. 9.3 for details.

Definition 9.2 (Trace-Simulation for HIOA)**.** *(adapted from* [LSV03]*) Given comparable HIOA* H_1,H_2, *a relation* $R \subseteq S_{H_1} \times S_{H_2}$ *is a* **trace-simulation relation** *for* H_1, H_2 *if and only if for all* $(p,q) \in R$ *a run* $\sigma_1 = p \xrightarrow{\alpha} p'$ *in* H_1 *implies a run* $\sigma_2 = q \xrightarrow{\beta} q'$ *in* H_2 *with* $\tau(\sigma_1)\downarrow_{Var_{E2}} = \tau(\sigma_2)\downarrow_{Var_{E2}}$ *and* $(p',q') \in R$. *We say that* H_2 **trace-simulates** H_1 *if there exists a trace-simulation relation R for* H_1, H_2 *such that for all* $p \in Init_{H_1}$ *there exists some* $q \in Init_{H_2}$ *such that* $(p,q) \in R$, *written as* $H_1 \preceq_{\text{tr}} H_2$.

While trace simulation is compositional [LSV03], it is for the general case computationally intractable, since functions are compared over an infinite number of time points. One has to reduce the problem to a finitary form, i.e., to a set of finitely many values that are to be checked. This could be achieved, e.g., by limiting the set of activities to a parameterized family of functions. Symbolic algorithms can then be applied if the symbolic sets of valuations can be described by some finite representation such as polyhedra, possibly in the form of an overapproximation. Our approach abstracts even further: We define simulation between hybrid I/O-automata based on their timed transition system semantics. That way the information on their continuous evolution is lost, but we obtain a computable framework for an interesting class of hybrid systems – mainly, linear hybrid automata.

Definition 9.3 (TTS-Simulation)**.** *A relation* $R \subseteq S_P \times S_Q$ *is a* **simulation relation** *for a pair of HLTS P,Q if and only if for all* $(p,q) \in R, \alpha \in \Sigma_P, p' \in S_P$ *holds:*

- $u\downarrow_{Var_{EQ}} = v\downarrow_{Var_{EQ}}$, *where* $p = (k,u)$ *and* $q = (l,v)$ *and*
- $p \xrightarrow{\alpha} p' \;\Rightarrow\; \exists q' \in S_Q : (q \xrightarrow{\alpha} q' \wedge (p',q') \in R)$.

A state q **simulates** *a state p if there exists some simulation relation R with* $(p,q) \in R$, *which is written as* $p \preceq q$. *Q* **simulates** *P, written as* $P \preceq Q$, *if and only if there exists a simulation relation R such that for all* $p \in Init_P$ *there exists a* $q \in Init_Q$ *such that* $(p,q) \in R$. *For HIOA* H_1,H_2, *let* $H_1 \preceq H_2$ *if and only if* $[\![H_1]\!] \preceq [\![H_2]\!]$. *We call this* **TTS-simulation**.

We define similarity and bisimulation, as before in Parts I and II, as equivalence relations based on asymmetric and symmetric simulation:

Definition 9.4 (Similarity, Bisimulation)**.** *Given HLTS P,Q Q is* **similar** *to P, written as $P \simeq Q$, if and only if $P \preceq Q$ and $Q \preceq P$. A relation $R \subseteq S_P \times S_Q$ is a* **bisimulation relation** *if and only if R witnesses $P \preceq Q$ and R^{-1} witnesses $Q \preceq P$. Q is* **bisimilar** *to P, written as $P \cong Q$, if and only if there exists such a relation. For HIOA H_1,H_2, let $H_1 \simeq H_2$ if and only if $[\![H_1]\!] \simeq [\![H_2]\!]$ and $H_1 \cong H_2$ if and only if $[\![H_1]\!] \cong [\![H_2]\!]$.*

We now show that TTS-simulation is consistent with trace-simulation in the sense that it is implied by it.

Proposition 9.5. *For any comparable HIOA H_1, H_2, $H_1 \preceq_{\mathrm{tr}} H_2$ implies $H_1 \preceq H_2$.*

Proof. Let R be a witnessing trace simulation relation between H_1 and H_2. We show that for all $(p,q) \in R$, where $p = (k,u)$ and $q = (l,v)$, the conditions of Def. 9.3 are fulfilled.

We start with the equality of the external variables. Consider any $(p,q) \in R$. There is always a run $\sigma_1 = p \xrightarrow{\varepsilon} p'$ possible in H_1, where $u = u'$. Because H_2 trace-simulates H_1, this implies a run $\sigma_2 = q \xrightarrow{\varepsilon} q'$ in H_2, with $\tau(\sigma_1)\downarrow_{Var_{E2}} = \tau(\sigma_2)\downarrow_{Var_{E2}}$. It immediately follows that $v\downarrow_{Var_{E2}} = v'\downarrow_{Var_{E2}} = u\downarrow_{Var_{E2}}$, which fulfills the first condition for TTS-simulation.

We now show that a transition in the TTS of H_1 implies a transition in the TTS of H_2, with both target states in R. Consider a transition $p \xrightarrow{\alpha}_{[\![H_1]\!]} p'$ in $[\![H_1]\!]$. It follows directly that there is a run $\sigma_1 = p \xrightarrow{\alpha'} p'$ in H_1, where $\alpha' = \alpha$ if $\alpha \in Lab_1 \cup \varepsilon$, and $\alpha' = (\alpha, f)$ if $\alpha \in \mathbb{R}^{\geq 0}$, where f is some activity of H_1. Because H_2 trace-simulates H_1, this implies a run $\sigma_2 = q \xrightarrow{\beta'} q'$ in H_2, with $\tau(\sigma_1)\downarrow_{Var_{E2}} = \tau(\sigma_2)\downarrow_{Var_{E2}}$ and $(p',q') \in R$. It follows from the definition of the TTS that there is a transition $q \xrightarrow{\beta}_{[\![H_2]\!]} q'$, where $\beta = \beta'$ if $\beta' \in Lab_2 \cup \varepsilon$, and $\beta = t$ if $\beta' = (t,g)$ for some $(t,g) \in \mathbb{R}^{\geq 0} \times Aty(Var_2)$. From $\tau(\sigma_1)\downarrow_{Var_{E2}} = \tau(\sigma_2)\downarrow_{Var_{E2}}$ follows that $\beta = \beta' = \alpha$ if $\alpha \in Lab_1 \cup \varepsilon$, and $\beta = t = \alpha$ otherwise. So from a transition in $[\![H_1]\!]$ we have deduced the existence of a transition in $[\![H_2]\!]$ with identical label, and their target states lie in R. Consequently, R is a TTS-simulation relation, and from the definition of trace simulation it follows that it is also a witness, which concludes the proof. □

In the following section we will examine how HLTS-based simulation can be applied in compositional reasoning.

9.2 Compositional Reasoning

In order for the simulation concept to be applied in a compositional analysis, it must hold under different contexts, i.e., when the automata are composed with other automata. This is captured by the concept of a precongruence, which has also been used extensively in Parts I and II. Before we can show that HLTS-simulation is a precongruence, we need the following lemma:

Lemma 9.6. *Given HLTS P,Q,S, S compatible with P and Q, let R be a simulation relation for $P \preceq Q$, and let $((k,u),(l,v)) \in R$. For every $x \in V(Var_P \cup Var_S)$ with $x{\downarrow}_{Var_P} = u$ there exists a state $y \in V(Var_Q \cup Var_S)$ with $y{\downarrow}_{Var_Q} = v$ and $y{\downarrow}_{Var_S} = x{\downarrow}_{Var_S}$.*

Proof. Consider a state $((k,u),(l,v)) \in R$ and an arbitrary valuation $x \in V(Var_P \cup Var_S)$ with $x{\downarrow}_{Var_P} = u$. By the definition of simulation, $Var_{OQ} = Var_{OP}$ and $Var_{IQ} \subseteq Var_{IP}$, and $u{\downarrow}_{Var_{OP}} = v_0{\downarrow}_{Var_{OP}}$, $u{\downarrow}_{Var_{IQ}} = v{\downarrow}_{Var_{IQ}}$. Consequently, it holds that

$$x_0{\downarrow}_{Var_{OP}} = v_0{\downarrow}_{Var_{OP}} = v_0{\downarrow}_{Var_{OQ}} \tag{9.1}$$

$$x_0{\downarrow}_{Var_{IQ}} = v_0{\downarrow}_{Var_{IQ}} \tag{9.2}$$

Let y be a valuation in $V(Var_Q \cup Var_S)$ such that $y{\downarrow}_{Var_S \setminus Var_Q} = x{\downarrow}_{Var_S \setminus Var_Q}$ and $y{\downarrow}_{Var_Q} = v$. It must be shown from the latter term that $y{\downarrow}_{Var_S \cap Var_Q} = x{\downarrow}_{Var_S \cap Var_Q}$. Then it follows that $y{\downarrow}_{Var_S} = x{\downarrow}_{Var_S}$. It holds that $y{\downarrow}_{Var_{IQ} \cap Var_S} = v{\downarrow}_{Var_{IQ} \cap Var_S}$, and with (9.2) it follows that $y{\downarrow}_{Var_{IQ} \cap Var_S} = x{\downarrow}_{Var_{IQ} \cap Var_S}$. From the definition of compatibility it follows that $Var_{SQ} \cap Var_S = Var_{OQ} \cap Var_S$. It holds that $y{\downarrow}_{Var_{OQ} \cap Var_S} = v{\downarrow}_{Var_{OQ} \cap Var_S}$, and with (9.1) it follows that $y{\downarrow}_{Var_{OQ} \cap Var_S} = x{\downarrow}_{Var_{OQ} \cap Var_S}$. Therefore $y{\downarrow}_{Var_S \cap Var_Q} = x{\downarrow}_{Var_S \cap Var_Q}$, which concludes the proof. □

Proposition 9.7. *Simulation of HLTS is a precongruence with respect to parallel composition, i.e., for any HLTS P,Q,S holds*

- *reflexivity: $P \preceq P$,*
- *transitivity: $P \preceq Q \wedge Q \preceq S \Rightarrow P \preceq S$, and*
- *invariance under composition: $P \preceq Q \Rightarrow P || S \preceq Q || S$.*

Proof. Let $p = (k,u)$, $q = (l,v)$, $s = (m,w)$ denote the states of P, Q and S.

Reflexivity: P is trivially comparable to P. $R = \{(p,p)\}$ is a witness that $\preceq$ is reflexive, since $p \xrightarrow{\alpha} p' \Rightarrow (p \xrightarrow{\alpha} p' \wedge (p',p') \in R)$. For all $(k,u,k',u') \in R, u{\downarrow}_{Var_{IP}} = u'{\downarrow}_{Var_{IP}}$ and $u{\downarrow}_{Var_{OP}} = u'{\downarrow}_{Var_{OP}}$. For any initial state p_0 in P there is a state $p' = p_0$ with $(p,p') \in R$.

Transitivity: First, we show that P is comparable to S. Since P is comparable to Q and Q is comparable to S, we have

$$Var_{OS} = Var_{OQ} = Var_{OP} \text{ and } Var_{IS} \subseteq Var_{IQ} \subseteq Var_{IP}, \tag{9.3}$$

as well as $\Sigma_S = \Sigma_Q = \Sigma_P$, and therefore P is comparable to S.

Let R_1 be a simulation relations for $P \preceq Q$, and R_2 for $Q \preceq S$. We show by structural induction that $R = \{(p,s) | \exists q : (p,q) \in R_1 \wedge (q,s) \in R_2\}$ is a simulation relation. Since $Q \preceq S$, there exists an initial state s_0 for any initial state q_0 of Q. Similarly, there exists an initial state q_0 for any initial state p_0 of P. Therefore, any initial state p_0 of P has a corresponding initial state s_0 in R. Consider a state $(p,s) \in R$. Then there exists a $q = (l,v)$ with $(p,q) \in R_1$ and $(q,s) \in R_2$. Therefore it holds that $u{\downarrow}_{Var_{IQ}} = v{\downarrow}_{Var_{IQ}}$ $\wedge u{\downarrow}_{Var_{OQ}} = v{\downarrow}_{Var_{OQ}}$ and $v{\downarrow}_{Var_{IS}} = w{\downarrow}_{Var_{IS}}$ $\wedge v{\downarrow}_{Var_{OS}} = w{\downarrow}_{Var_{OS}}$. With (9.3) follows that

$u{\downarrow}_{Var_{IS}} = w{\downarrow}_{Var_{IS}} \wedge u{\downarrow}_{Var_{OS}} = w{\downarrow}_{Var_{OS}}$. Since R_1 is a simulation relation, a transition $p \xrightarrow{\alpha} p'$ implies that there exists a q' with $q \xrightarrow{\alpha} q'$ and $(p',q') \in R_1$. Similarly, there exists a s' with $(q',s') \in R_2$, and therefore $(p',s') \in R$.

Invariance under Composition: Let P, Q and S be hybrid labeled transition systems, and let $p = (k,u)$, $q = (l,v)$, $s = (m,w)$ denote their respective hybrid states. First, we show that $P||S$ is comparable to $Q||S$. Since P is comparable to Q, it holds that

$$Var_{OQ} = Var_{OP} \text{ and } Var_{IQ} \subseteq Var_{IP}, \tag{9.4}$$

as well as $\Sigma_Q = \Sigma_P$. Therefore $Var_{OQ} \cup Var_{OS} = Var_{OP} \cup Var_{OS}$ and $Var_{IQ} \cup Var_{IS} \subseteq Var_{IP} \cup Var_{IS}$. Since $Var_{IQ} \cap Var_{OQ} = Var_{IS} \cap Var_{OS} = \emptyset$, $Var_{IQ||S} = (Var_{IQ} \cup Var_{IS}) \setminus (Var_{OQ} \cup Var_{OS}) = (Var_{IQ} \setminus Var_{OS}) \cup (Var_{IS} \setminus Var_{OQ})$. Similarly, $Var_{IP||S} = (Var_{IP} \cup Var_{IS}) \setminus (Var_{OP} \cup Var_{OS}) = (Var_{IP} \cup Var_{IS}) \setminus (Var_{OQ} \cup Var_{OS}) = (Var_{IP} \setminus Var_{OS}) \cup (Var_{IS} \setminus Var_{OQ})$. Consequently, $Var_{IQ||S} \subseteq Var_{IP||S}$, and $P||S$ is comparable to $Q||S$.

Consider a simulation relation R_0 for $P \preceq Q$. Let a state in $P||S$ be denoted by (k,m,x), where $k \in Loc_P$, $m \in Loc_S$ and $x \in V(Var_P \cup Var_S)$. Similarly a state in $Q||S$ is denoted by (l,m,y). We show by structural induction that

$$R = \{((k,m,x),(l,m,y)) \mid \exists u,v : ((k,u),(l,v)) \in R_0, \\ x{\downarrow}_{Var_P} = u, y{\downarrow}_{Var_Q} = v, x{\downarrow}_{Var_S} = y{\downarrow}_{Var_S}\}$$

is a simulation relation.

It must be shown that for any initial state (k_0,m_0,x_0) there exists an initial state (l_0,m_0,y_0) with $((k_0,m_0,x_0),(l_0,m_0,y_0)) \in R$. By the definition of parallel composition, it holds for (k_0,x_0) that there exists a u_0 with $(k_0,u_0) \in Init_P$ and $u_0 = x_0{\downarrow}_{Var_P}$. Since $P \preceq Q$, this implies that there exists some initial state $(l_0,v_0) \in Init_Q$ for which it holds that $((k_0,u_0),(l_0,v_0)) \in R_0$. From Lemma 9.6 it follows that there exists a $y_0 \in V(Var_Q \cup Var_S)$ such that $y_0{\downarrow}_{Var_S} = x_0{\downarrow}_{Var_S}$ and $y_0{\downarrow}_{Var_Q} = v_0$. Consequently, $((k_0,m_0,x_0),(l_0,m_0,y_0)) \in R$, and furthermore $(l_0,y_0{\downarrow}_{Var_Q}) \in Init_Q$ and $(m_0,y_0{\downarrow}_{Var_S}) \in Init_S$, which by the definition of parallel composition implies that $(l_0,m_0,y_0) \in Init_{Q||S}$.

Now we show that for any $((k,m,x),(l,m,y)) \in R$ holds $x{\downarrow}_{Var_{I(Q||S)}} = y{\downarrow}_{Var_{I(Q||S)}}$ and $x{\downarrow}_{Var_{O(Q||S)}} = y{\downarrow}_{Var_{O(Q||S)}}$. Since $((k,m,x),(l,m,y)) \in R$, there exists $u = x{\downarrow}_{Var_P}$ and $v = y{\downarrow}_{Var_Q}$ with $((k,u),(l,v)) \in R_0$. Consequently, $x{\downarrow}_{Var_{OQ}} = v{\downarrow}_{Var_{OQ}} = y{\downarrow}_{Var_{OQ}}$ and $x{\downarrow}_{Var_{IQ} \setminus Var_{OQ}} = v{\downarrow}_{Var_{IQ} \setminus Var_{OQ}} = y{\downarrow}_{Var_{IQ} \setminus Var_{OQ}}$. Since $x{\downarrow}_{Var_S} = y{\downarrow}_{Var_S}$ it follows directly that $x{\downarrow}_{Var_{OS}} = y{\downarrow}_{Var_{OS}}$ and $x{\downarrow}_{Var_{IS} \setminus Var_{OQ}} = y{\downarrow}_{Var_{IS} \setminus Var_{OQ}}$.

Consider some $((k,m,x),(l,m,y)) \in R$. It must be shown that for any transition $(k,m,x) \xrightarrow{\alpha}_{P||S} (k',m',x')$ there exists a transition $(l,m,y) \xrightarrow{\alpha}_{Q||S} (l',m',y')$ with $((k',m',x'),(l',m',y')) \in R$.

(i) $\alpha \in \Sigma_P$: Then by the definition of composition $(k,m,x) \xrightarrow{\alpha}_{P||S} (k',m',x')$ implies $(k,u) \xrightarrow{\alpha}_P (k',u')$ with $u = x{\downarrow}_{Var_P}$ and $u' = x'{\downarrow}_{Var_P}$, as well as a transition $(m,w) \xrightarrow{\beta}_S (m',w')$ with $w = x{\downarrow}_{Var_S}$, $w' = x'{\downarrow}_{Var_S}$ and $\beta = \alpha$ if $\alpha \in \Sigma_P \cap \Sigma_S$,

and $\beta = \varepsilon$ otherwise. With $((k,u),(l,v)) \in R_0$ the transition in P implies a transition $(l,v) \xrightarrow{\alpha}_Q (l',v')$ such that $((k',u'),(l',v')) \in R_0$. From Lemma 9.6 it follows that there exists a $y' \in V(Var_Q \cup Var_S)$ such that $y'\downarrow_{Var_S} = x'\downarrow_{Var_S}$ and $y'\downarrow_{Var_Q} = v'$. By definition of parallel composition, there exists a transition $(l,m,y) \xrightarrow{\alpha}_{Q||S} (l',m',y')$. Consequently, $((k',m',x'),(l',m',y')) \in R$.

(ii) $\alpha \in \Sigma_S \backslash \Sigma_P$: Then by the definition of composition $(k,m,x) \xrightarrow{\alpha} (k',m',x')$ implies $(m,w) \xrightarrow{\alpha} (m',w')$ with $w = x\downarrow_{Var_S}$, $w' = x'\downarrow_{Var_S}$ and $x\downarrow_{Var_{CP}} = x'\downarrow_{Var_{CP}}$, and an environment transition $(k,u) \xrightarrow{\varepsilon}_P (k',u')$ with $u = x\downarrow_{Var_P}$ and $u' = x'\downarrow_{Var_P}$. Therefore $((k,u),(l,v)) \in R_0$ implies that (l,v') exists with $(l,v) \xrightarrow{\varepsilon}_Q (l',v')$ and $((k',u'),(l,v')) \in R_0$. From Lemma 9.6 it follows that there exists a $y' \in V(Var_Q \cup Var_S)$ such that $y'\downarrow_{Var_S} = x'\downarrow_{Var_S}$ and $y'\downarrow_{Var_Q} = v'$. Consequently, it holds that $((k,m',x'),(l,m',y')) \in R$.

□

Corollary 9.8. *From Prop. 9.7 and the definition of HLTS-simulation of HIOA it follows that HLTS-simulation of HIOA is a preorder, i.e., reflexive and transitive.*

The composition of two HIOA has a semantics that differs from the composition of their TTS, which is why invariance under composition does not apply to simulation between HIOA. This also voids compositionality as it was applicable to LTS in Part I and hybrid automata without shared variables in Part II. In the following we will obtain a weaker compositionality rule by showing that there is a simulation relation between the two.

The composition of TTS is a conservative abstraction of the hybrid automaton behavior, i.e., if there is a transition in the composed hybrid automaton then there is also one in the composition of the TTS:

Lemma 9.9. *For any compatible HIOA H_1, H_2 and label $\alpha \in Lab_1 \cup Lab_2 \cup \varepsilon$ holds:*

$$((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1||H_2]\!]} ((l_1',l_2'),v') \quad \Leftrightarrow \quad ((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1]\!]||[\![H_2]\!]} ((l_1',l_2'),v')$$

and for all $\alpha \in \mathbb{R}^{\geq 0}$ holds:

$$((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1||H_2]\!]} ((l_1',l_2'),v') \quad \Rightarrow \quad ((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1]\!]||[\![H_2]\!]} ((l_1',l_2'),v').$$

Proof. We show that the transitions are identical for all except the timed transitions, in which case there is only implication.

- discrete transitions with $\alpha = Lab_1 \cap Lab_2$:
  $((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1||H_2]\!]} ((l_1',l_2'),v') \Leftrightarrow$ (Def. of TTS) $\exists \mu$: $(l_1,l_2) \xrightarrow{\alpha,\mu}_{H_1||H_2} (l_1',l_2')$ with $(v,v') \in \mu$, $v \in Inv(l_1,l_2)$, $v' \in Inv(l_1',l_2')$

$\Leftrightarrow$ (Def. of PC of HIOA) $\exists \mu_i$: $l_i \xrightarrow{\alpha,\mu_i}_{H_i} l_i', (v{\downarrow}_{Var_i}, v'{\downarrow}_{Var_i}) \in \mu_i$ and $v{\downarrow}_{Var_i} \in Inv_i(l_i)$, $v'{\downarrow}_{Var_i} \in Inv_i(l_i')$, $i = 1,2$
$\Leftrightarrow$ (Def. of TTS) $(l_i, v{\downarrow}_{Var_i}) \xrightarrow{\alpha}_{[\![H_i]\!]} (l_i', v'{\downarrow}_{Var_i})$, $i = 1,2$
$\Leftrightarrow$ (Def. of PC of HLTS) $((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1]\!] \,||\, [\![H_2]\!]} ((l_1',l_2'),v')$

- discrete transitions with $\alpha \in Lab_1 \backslash Lab_2$ ($\alpha \in Lab_2 \backslash Lab_1$ is symmetric):
  $((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1 \,||\, H_2]\!]} ((l_1',l_2'),v') \Leftrightarrow$ (Def. of TTS) $\exists \mu$: $(l_1,l_2) \xrightarrow{\alpha,\mu}_{H_1 \,||\, H_2} (l_1',l_2')$ with $(v,v') \in \mu$, $v \in Inv(l_1,l_2)$, $v' \in Inv(l_1',l_2')$
  $\Leftrightarrow$ (Def. of PC of HIOA) $\exists \mu_1$: $l_1 \xrightarrow{\alpha,\mu_1}_{H_1} l_1', (v{\downarrow}_{Var_1}, v'{\downarrow}_{Var_1}) \in \mu_1$ and $v{\downarrow}_{Var_1} \in Inv_1(l_1)$, $v'{\downarrow}_{Var_1} \in Inv_1(l_1')$, $v{\downarrow}_{Var_{C2}} = v'{\downarrow}_{Var_{C2}}$, $v{\downarrow}_{Var_2}, v'{\downarrow}_{Var_2} \in Inv_2(l_2)$, $l_2 = l_2'$
  $\Leftrightarrow$ (Def. of TTS) $(l_1, v{\downarrow}_{Var_1}) \xrightarrow{\alpha}_{[\![H_1]\!]} (l_1', v'{\downarrow}_{Var_1})$, $(l_2, v{\downarrow}_{Var_2}) \xrightarrow{\varepsilon}_{[\![H_2]\!]} (l_2', v'{\downarrow}_{Var_2})$
  $\Leftrightarrow$ (Def. of PC of HLTS) $((l_1,l_2),v) \xrightarrow{\alpha}_{[\![H_1]\!] \,||\, [\![H_2]\!]} ((l_1',l_2'),v')$

- environment transitions, $\alpha = \varepsilon$:
  $((l_1,l_2),v) \xrightarrow{\varepsilon}_{[\![H_1 \,||\, H_2]\!]} ((l_1,l_2),v')$
  $\Leftrightarrow$ (Def. of TTS) $v{\downarrow}_{Var_{C1} \cup Var_{C2}} = v'{\downarrow}_{Var_{C1} \cup Var_{C2}}$, $v \in Inv(l_1,l_2)$, $v' \in Inv(l_1',l_2')$
  $\Leftrightarrow$ (Def. of PC of HIOA and associativity of projection) $v{\downarrow}_{Var_{Ci}} = v'{\downarrow}_{Var_{Ci}}$ and $v{\downarrow}_{Var_i} \in Inv_i(l_i)$, $v'{\downarrow}_{Var_i} \in Inv_i(l_i')$, $i = 1,2$
  $\Leftrightarrow$ (Def. of TTS) $(l_i, v{\downarrow}_{Var_i}) \xrightarrow{\varepsilon}_{[\![H_i]\!]} (l_i, v'{\downarrow}_{Var_i})$, $i = 1,2$
  $\Leftrightarrow$ (Def. of PC of HLTS) $((l_1,l_2),v) \xrightarrow{\varepsilon}_{[\![H_1]\!] \,||\, [\![H_2]\!]} ((l_1,l_2),v')$

- timed transitions, $\alpha = t \in \mathbb{R}$:
  $((l_1,l_2),v) \xrightarrow{t}_{[\![H_1 \,||\, H_2]\!]} ((l_1',l_2'),v') \Rightarrow$ (Def. of TTS) $\exists f(t) \in Act((l_1,l_2))$, $f(0) = v$, $f(t) = v'$ and $\forall t', 0 \le t' \le t : f(t) \in Inv(l_1,l_2)$, $(l_1,l_2) = (l_1',l_2')$
  $\Rightarrow$ (Def. of PC of HIOA) $f{\downarrow}_{Var_1} \in Act_1(l_1)$ and $f{\downarrow}_{Var_2} \in Act_2(l_2)$, $f(0) = v$, $f(t) = v'$ and $\forall t', 0 \le t' \le t : f(t){\downarrow}_{Var_1} \in Inv_1(l_1) \wedge f(t){\downarrow}_{Var_2} \in Inv_2(l_2)$
  $\Rightarrow$ (Def. of TTS) $(l_i, v{\downarrow}_{Var_i}) \xrightarrow{t}_{[\![H_i]\!]} (l_i', v'{\downarrow}_{Var_i})$
  $\Rightarrow$ (Def. of PC of HLTS) $((l_1,l_2),v) \xrightarrow{t}_{[\![H_1]\!] \,||\, [\![H_2]\!]} ((l_1',l_2'),v')$

□

Lemma 9.9 prompts a simple corollary to identify HIOA for which the HLTS semantics and parallel composition are commutative. Since this immediately leads to compositionality on the HIOA level, we call automata that fulfill this condition ***TTS-compositional***.

Corollary 9.10. *For any compatible HIOA H_1, H_2 holds $[\![H_1 \,||\, H_2]\!] = [\![H_1]\!] \,||\, [\![H_2]\!]$ if for any $t \in \mathbb{R}$ the existence of transitions $(l_i, v{\downarrow}_{Var_i}) \xrightarrow{t}_{[\![H_i]\!]} (l_i, v{\downarrow}_{Var_i})$ for $i = 1,2$ implies a transition $((l_1,l_2),v) \xrightarrow{t}_{[\![H_1 \,||\, H_2]\!]} ((l_1',l_2'),v')$.*

The overapproximation that results from composing the TTSs can be formally expressed as a simulation:

Proposition 9.11. *For any H_1, H_2, $[\![H_1 \,||\, H_2]\!] \preceq [\![H_1]\!] \,||\, [\![H_2]\!]$.*

Proof. From the definitions of parallel composition it follows directly that variables, alphabet and states are identical. With Lemma 9.9, a transition in $[\![H_1||H_2]\!]$ also is a transition in $[\![H_1]\!]||[\![H_2]\!]$. Therefore $R = \{(((l_1,l_2),v),((l_1,l_2),v))\}$ is a witness for simulation. □

Recall that the compositional rule for systems without shared variables, Theorem 6.9, was given by

$$H_1 \preceq G_1 \wedge H_2 \preceq G_2 \;\Rightarrow\; H_1||H_2 \preceq G_1||G_2.$$

With the shared variables in HIOA, this implication is no longer valid, and we must show in addition that the right hand side supports the overapproximation introduced by the HLTS-simulation. This leaves us with a weaker form of compositionality for HIOA:

Proposition 9.12 (Weak Compositionality)**.** *For any HIOA H_1,H_2,G_1,G_2 with $H_1 \preceq G_1$, $H_2 \preceq G_2$ and $[\![G_1]\!] \,||\, [\![G_2]\!] \preceq [\![Gs_1 \,||\, G_2]\!]$ it follows that $H_1||H_2 \preceq G_1||G_2$.*

Proof. According to Prop. 9.7, all the precongruence properties, and therefore compositionality, hold for the TTS, which leads to $[\![H_1]\!]||[\![H_2]\!] \preceq [\![G_1]\!]||[\![G_2]\!]$. With the hypothesis and transitivity we get $[\![H_1]\!]||[\![H_2]\!] \preceq [\![G_1 \,||\, G_2]\!]$. According to Prop. 9.11 holds $[\![H_1 \,||\, H_2]\!] \preceq [\![H_1]\!] \,||\, [\![H_2]\!]$ and with transitivity follows $[\![H_1 \,||\, H_2]\!] \preceq [\![G_1 \,||\, G_2]\!]$. □

We will now use Corollary 9.10 to identify classes of HIOA for which TTS-simulatin is compositional. The definition of HIOA permits an HIOA to restrict and change input variables. In seeking strong compositionality we define what it means for an HIOA to have no influence on a set of variables, e.g., its inputs:

Definition 9.13 (Unrestricted Variables)**.** *A hybrid I/O-automaton has a* **unrestricted invariants** *with respect to a set of variables Var' if for all $v \in Inv(l)$, $v' \in V(Var)$, $v'{\downarrow}_{Var \setminus Var'} = v{\downarrow}_{Var \setminus Var'}$: $v' \in Inv(l)$. It has* **unrestricted activities** *with respect to Var' if any activity over Var' is accepted, i.e., for all $g \in Ats(Var)$, $f \in Act(l)$ there exists a $f' \in Act(l)$ with $g{\downarrow}_{Var'} = f'{\downarrow}_{Var'}$ and $f{\downarrow}_{Var \setminus Var'} = f'{\downarrow}_{Var \setminus Var'}$. If both conditions are fulfilled, it is called* **unrestricted** *in Var'.*

The first condition prevents the variables to trigger any events. Changes in the system must be caused by other variables, such as a sampling clock. The second condition effectively prohibits the use of the variables in the definition of the activities. For unrestricted inputs, this prevents inputs of the form $\dot{x} = f(u)$. In such a case, the only way for the automaton to react to its inputs is by an internally triggered discrete transition, whose guard can depend on the inputs. Obviously, unrestricted inputs are a severe restriction to make, but it also has a high merit: There is no excess behavior when composing the TTS, and so the system is fully open to compositional reasoning:

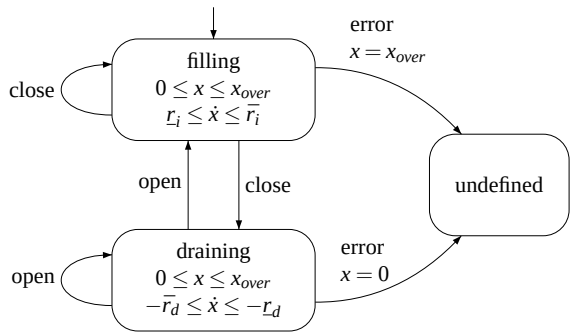
Theorem 9.14. *For any compatible HIOA H_1, H_2, where for $(i,j) \in \{(1,2),(2,1)\}$ holds that H_i has unrestricted variables $Var_{Ii} \cap Var_{Ej}$, it also holds that $[\![H_1 \,||\, H_2]\!] = [\![H_1]\!] \,||\, [\![H_2]\!]$.*

Proof. We show that H_1,H_2 fulfill conditions of Corollary 9.10. Since for both H_i the activities for the common input variables are unrestricted and the control variables are disjoint, there exist activities $f_i' \in Act_i(l_i)$ with $f_i'{\downarrow}_{Var_{Cj}} = f_j{\downarrow}_{Var_{Cj}}$ and $f_i'{\downarrow}_{Var_{Ci}} = f_i{\downarrow}_{Var_{Ci}}$ for $(i,j) \in \{(1,2),(2,1)\}$. Therefore we can pick the activities from the variables of both and obtain a joint activity:Let $g \in Ats(Var)$ be such that $g{\downarrow}_{Var_{Ci}} = f_i{\downarrow}_{Var_{Ci}}$, $g{\downarrow}_{Var_{Ii} \setminus Var_{Ej}} = f_i{\downarrow}_{Var_{Ii} \setminus Var_{Ej}}$ for $(i,j) \in \{(1,2),(2,1)\}$. For all variables, the activity g is either identical with the activities of H_i, or H_i has unrestricted activities and invariants. Therefore $g \in Act_{H_1||H_2}(l_1,l_2)$ and for all $t', 0 \leq t' \leq t$ holds $g(t) \in Inv_{H_1||H_2}(l_1,l_2)$. Consequently, there is a transition $((l_1,l_2),v) \xrightarrow{t}_{[\![H_1||H_2]\!]} ((l_1',l_2'),v')$. □

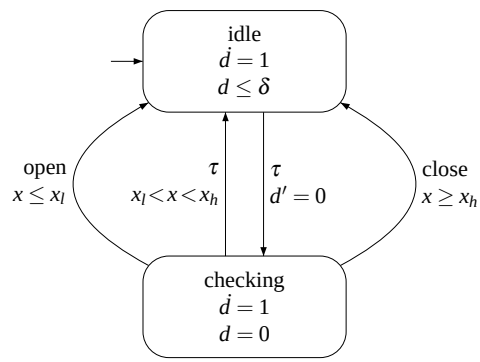
A class of HIOA with unrestricted input variables is given by digitally controlled processes. The continuous inputs are unrestricted and sampled, so that the change of the inputs in between sampling points is irrelevant. If the controller has continuous outputs they usually change only at discrete sampling intervals. These can by modeled by a zero-order-hold element. If this zero-order-hold filter is included in the process model, both inputs and outputs of the controller are sampled. The following is an example of a controller with unrestricted inputs:

Example 9.1. *Consider a tank level monitoring system consisting of a tank with continuous outflow, a discrete inlet valve (modeled as part of the tank), and a controller. The tank is modeled as a HIOA P_1, shown in Fig. 9.1(b). Its inlet valve is operated by the controller via the labels* open *and* close *that represent the opening and closing of the inlet valve. The level x of the tank changes at a rate $\underline{r}_i \leq \dot{x} \leq \overline{r}_i$ if the valve is open, and at a rate of $-\overline{r}_d \leq \dot{x} \leq -\underline{r}_d$ if it is closed. The location "undefined" represents states that were excluded from the model and are only reachable by transitions with label "error". P_1 has the state and output variable x, and no input variable. The controller, modeled by P_2 in Fig. 9.1(a), is triggered by the timer d every δ seconds to check the level of the tank, and instantly decides whether to open the valve, close it or do nothing, and returns to the idle state. P_2 has the input variable x, the control variable d and no output variable. The goal of the verification will be to show that the tank level stays within the limits $x_m \leq x \leq x_M$ and that the model remains within the modeling bounds, i.e., produces no "error"-transitions.*

Example 9.2. *Consider the tank level monitoring system from Ex. 9.1. The specification Q for the tank-controller system is that the level in the tank stays between x_m and x_M. This invariant can be specified using a single location, as shown in Fig. 9.2. Self-loops allow the labels τ, "open" and "close" at any time. The label "error" never occurs, so that states that are beyond the model boundaries must not be reachable. Q has the state and output variable x, and no input variables.*



(a) Tank P_1



(b) Controller P_2

Figure 9.1: Tank level monitoring system

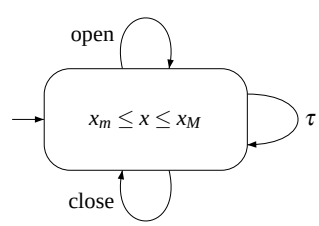


Figure 9.2: Specification Q of the composed system

There is another interesting class of TTS-compositional hybrid systems: the Linear Hybrid Automata from Sect. 5.6, as long as they have convex, unrestricted invariants with respect to their common variables. The reason is that the class of activities of LHA allows us to generate an activity once the existence of any activity is guaranteed:

Lemma 9.15. (adapted from [Ho95]) *Let l be a location of any linear HIOA with a convex invariant $Inv(l)$, and $v, v' \in Inv(l)$ be any valuations inside it. If there exists an activity $f \in Act(l)$ and a duration $\delta \in \mathbb{R}^{\geq 0}$ such that $f(0) = v$, $f(\delta) = v'$ and $\forall t', 0 \leq t' \leq \delta : f(t') \in Inv(l)$ then there also exists $f'(t) = v + t/\delta(v' - v) \in Act(l)$ and $\forall t', 0 \leq t' \leq \delta : f'(t') \in Inv(l)$.*

Simply put, a straight line between two points is always an activity as long as the invariant is convex. The necessity of convexity was already illustrated by Ex. 8.2. Lemma 9.15 allows us to infer an activity of a hybrid automaton from the mere existence of a transition in its TTS. We can use this in compositional reasoning to conclude that if the TTS of two LHA have transitions with the same source and target valuations, then both share a common activity that connects the valuations, and therefore this activity also exists in their parallel composition:

Theorem 9.16. *For any linear HIOA H_1, H_2 with convex invariants holds that*

$$[\![H_1 \,||\, H_2]\!] = [\![H_1]\!] \,||\, [\![H_2]\!].$$

Proof. We show that the conditions of Corollary 9.10 are fulfilled. By definition of the TTS, the transitions $(l_i, v_i) \xrightarrow{\delta}_{[\![H_i]\!]} (l_i, v_i')$ imply that there exists activities f_i that respect the invariants. With Lemma 9.15 it follows that

$$f_i'(t) = v + t/\delta(v' - v), \text{ for } i = 1, 2,$$

are also valid activities. Since $f_1' \downarrow_{Var_1 \cap Var_2} = f_2' \downarrow_{Var_1 \cap Var_2}$, there exists an activity $f \in Ats(Var_1 \cup Var_2)$ with $f \downarrow_{Var_i} = f_i$ for $i = 1, 2$. By definition of parallel composition, f respects the invariant of $H_1 || H_2$ in the location (l_1, l_2) and $f \in Act((l_1, l_2))$. Consequently, f is a witness for the transition $((l_1, l_2), v) \xrightarrow{\delta}_{[\![H_1 || H_2]\!]} ((l_1, l_2), v')$. □

We can now show that linear HIOA with convex invariants are compositional:

Corollary 9.17 (Compositionality)**.** *For any HIOA P_1,P_2 and linear HIOA Q_1,Q_2, Q_1,Q_2 with convex invariants, for which holds $P_1 \preceq Q_1$, $P_2 \preceq Q_2$ it also holds that $P_1 || P_2 \preceq Q_1 || Q_2$.*

Proof. The conclusion follows directly from Prop. 9.12 and Theorem 9.16. □

Given that any HIOA can be approximated arbitrarily close with linear HIOA [HHWT98], linear HIOA allow us to perform compositional reasoning for any HIOA. This even motivates the use of hybrid automata in the analysis of purely continuous systems: They can be approximated with linear HIOA, which are then analyzed compositionally.

9.3 Related Work

Simulation While simulation that abstracts from internal actions, like in [LSV03], is useful in practise, it renders the analysis more difficult and complicates the proofs. Since our focus is on the continuous interaction, we keep the simple concept of synchronization labels that are accessible to all automata. Internal actions, which are not restricted by the specification, can be modeled by adding self-loops to the specification for those labels.

Compositionality A compositional framework for abstraction of hybrid control systems was presented [TPL04]. Abstraction is defined as the existence of a mapping between sets of generalized action maps over the states. In our context it corresponds to trace inclusion. The authors show that simulation and bisimulation is compositional. A very general discussion about compositionality is carried out in [dAH01]. The authors separate the interface of modules from the definition of its behavior and, using game theory, infer general properties that abstractions must fulfill in order to support compositional reasoning. In [BS98] the authors discuss different composition operators and their effect on deadlock-freedom and maximal progress. Compositional refinement is shown for the hierarchical modeling language CHARON in [AGLS01]. Refinement is defined as trace inclusion. The structure and semantics of the hierarchical constraints on flows are defined in a way that compositionality is guaranteed. The existence of continuous dynamics in the composition of hybrid systems is examined in [SL02], where a procedure for computing the *inner viability kernel* is presented that allows one to conclude that the activities of the composed system are non-empty. The inner viability kernel has to be non-empty for the approach to work. Our definition of unrestricted inputs is closely related to the notion of *oblivious* HIAO in [LSV03].

Chapter 10

Assume-Guarantee Reasoning

Assume-guarantee reasoning aims at deducing the behavior of a composed system from an analysis of parts of the system under assumptions about the rest of the system. Consider a system with hybrid automata $P = P_1 \,||\, P_2$ with specification $Q = Q_1 \,||\, Q_2$. The goal is to show that $P_1 \,||\, P_2 \preceq Q_1 \,||\, Q_2$, which by definition is equal to showing that $[\![P_1 \,||\, P_2]\!] \preceq [\![Q_1 \,||\, Q_2]\!]$ for their timed transition systems.

In Sect. 4 we introduced non-circular and circular assume-guarantee rules for labeled transition systems, which we will in the following adapt to hybrid systems with shared variables. As in the previous chapter, we must operate with timed transitions systems since TTS-simulation is not compositional for general hybrid I/O-automata. While in cases with continuous input this can lead to a prohibitively large overapproximation, it does enable the proof for some interesting examples and applications. No overapproximation takes place in the special cases identified in Sect. 9.2, i.e., hybrid automata with unrestricted inputs and linear hybrid automata with convex invariants.

In the following section, we formulate a rule for non-circular reasoning. It uses the weak compositionality rule from Sect. 9.2. In Sect. 10.2, we propose an adaptation of the assume/guarantee rule of Part I to HIOA. The assume/guarantee conditions are enhanced to ensure that independent transitions of one automaton do not violate the invariants of the other automata. A summary of related work can be found in Sect. 10.3.

10.1 Non-circular Assume-Guarantee Reasoning

Non-circular assume-guarantee reasoning occurs if the abstraction of one automaton serves as the guarantee to another, yielding a triangular structure:

$$\frac{\begin{array}{rcl} [\![P_1]\!] & \preceq & [\![Q_1]\!] \\ [\![Q_1]\!] \,||\, [\![P_2]\!] & \preceq & [\![Q_1 \,||\, Q_2]\!] \end{array}}{P_1 \,||\, P_2 \preceq Q_1 \,||\, Q_2}. \tag{10.1}$$

The proof is straightforward using the precongruence properties of simulation for HLTS and Prop. 9.11: $P_1 \preceq Q_1$ implies $[\![P_1]\!] \,||\, [\![P_2]\!] \preceq [\![Q_1]\!] \,||\, [\![P_2]\!]$ due to the invariance of the simulation of HLTS under composition. Through transitivity it follows

from $[\![Q_1]\!] \,||\, [\![P_2]\!] \preceq [\![Q_1 \,||\, Q_2]\!]$ that $[\![P_1]\!] \,||\, [\![P_2]\!] \preceq [\![Q_1 \,||\, Q_2]\!]$. With Prop. 9.11 and applying transitivity we get $[\![P_1 \,||\, P_2]\!] \preceq [\![Q_1 \,||\, Q_2]\!]$, which by definition of simulation yields $P_1 \,||\, P_2 \preceq Q_1 \,||\, Q_2$.

10.2 Circular Assume/Guarantee Reasoning

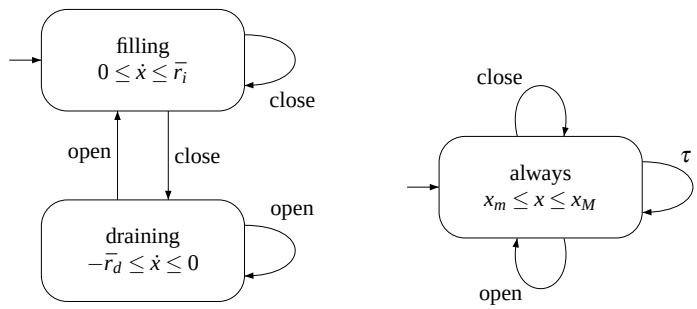
As discussed in the previous parts of this thesis, a circular assume/guarantee proof is only sound if additional conditions, in the following called *A/G conditions*, ensure that Q_1 and Q_2 do not block transitions in their composition that are enabled for the composition of P_1 and P_2. We recall the basic structure of a circular assume/guarantee proof:

$$\frac{\begin{array}{ccc} [\![P_1]\!] \,||\, [\![Q_2]\!] & \preceq & [\![Q_1 \,||\, Q_2]\!] \\ [\![Q_1]\!] \,||\, [\![P_2]\!] & \preceq & [\![Q_1 \,||\, Q_2]\!] \\ \multicolumn{3}{c}{\textit{A/G conditions}} \end{array}}{P_1 \,||\, P_2 \;\preceq\; Q_1 \,||\, Q_2}. \tag{10.2}$$

The following example shall illustrate how efficiently assume/guarantee reasoning can simplify proofs by reducing the number of continuous variables, and abstracting from dynamic details if only the range of variables is of importance:

Example 10.1. *Consider the tank level monitoring system from Ex. 9.1. To verify the global specification Q from Fig. 9.2 using assume-guarantee reasoning, specifications Q_i are created manually for each subsystem. It is then checked that their composition guarantees Q, i.e., that $Q_1 || Q_2 \preceq Q$. The specification Q_2 for the controller, shown in Fig. 10.1(b), simply requires that the controller keeps the tank level inside the bounds x_m and x_M. Q_2 has no state variables and x as an input variable. Note that Q_2 represents the function of the controller, and is a proper specification, i.e., it has nothing to do with the controller implementation P_2. Apart from allowing to abstract from implementation details such as the timer d, a proper specification has the advantage that it doesn't have to be reinvented whenever the implementation changes. The specification Q_1 for the tank, see Fig. 10.1(a), is a simplified version of P_1. The inflow and outflow rate are overapproximated and the invariants as well as the location "undefined" are omitted. The essential information that guarantees the functioning of the controller within the A/G-reasoning is that the level rises after opening of the valve, and falls after closing. The label "error" is in Lab_{Q_1}, but is never allowed. Q_1 has the state and output variable x, and no input variables. Note that neither Q_1 nor Q_2 are conservative overapproximations of P_1 and P_2.*

To simplify the notation, we write for valuations $u{\downarrow_P}$ instead of $u{\downarrow_{Var_P}}$, $u{\downarrow_{IP}}$ instead of $u{\downarrow_{Var_{IP}}}$ etc. The A/G-conditions for HIOA are given by the following theorem:



(a) Tank spec. Q_1 (b) Controller spec. Q_2

Figure 10.1: Modular specifications for A/G-reasoning

Theorem 10.1 (A/G-simulation)**.** *Consider hybrid I/O-automata P_i, Q_i, P_i comparable to Q_i, for $i = 1,2$ and with*

$$[\![P_1]\!] \,||\, [\![Q_2]\!] \;\preceq\; [\![Q_1 \,||\, Q_2]\!] \quad \textit{and} \tag{10.3}$$

$$[\![Q_1]\!] \,||\, [\![P_2]\!] \;\preceq\; [\![Q_1 \,||\, Q_2]\!]. \tag{10.4}$$

If there exist simulation relations R_1 for (10.3) and R_2 for (10.4), the relation

$$\begin{aligned} R = \{((k_1,k_2,x),(l_1,l_2,z)) | \exists y_i, \hat{l}_j, \hat{z}_i : ((k_i,l_j,y_i),(l_i,\hat{l}_j,\hat{z}_i)) \in R_i, \\ y_i{\downarrow}_{P_i} = x{\downarrow}_{P_i}, y_i{\downarrow}_{Q_j} = z{\downarrow}_{Q_j}, \hat{z}_i{\downarrow}_{Q_i} = z{\downarrow}_{Q_i} \;\; \textit{for } (i,j) \in \{(1,2),(2,1)\}\} \end{aligned} \tag{10.5}$$

is a simulation relation for $P_1 \,||\, P_2 \preceq Q_1 \,||\, Q_2$ (but not necessarily a witness), if for all $((k_1,k_2,x),(l_1,l_2,z)) \in R$, a transition $(k_1,k_2,x) \xrightarrow{\alpha}_{P_1||P_2} (k_1',k_2',x')$ implies that there exists a transition

$$(l_i, z{\downarrow}_{Q_i}) \xrightarrow{\beta}_{[\![Q_i]\!]} (l_i', z'{\downarrow}_{Q_i}),\; z'{\downarrow}_{P_j \cap Q_i} = x'{\downarrow}_{P_j \cap Q_i}, (i,j) \in \{(1,2),(2,1)\} \tag{10.6}$$

and $\beta = \varepsilon$ if $\alpha \in \Sigma_{P_j} \setminus \Sigma_{P_i}$ or $\beta = \alpha$ if $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$.

Proof. Let $\Sigma_i = \Sigma_{Pi}$. The proof shows that R is a simulation relation. Note that while y in (10.5) is uniquely defined, $\hat{z}_j$ is undefined in the variables $Var_{Q_i} \setminus Var_{Q_j}$ for $(i,j) \in \{(1,2),(2,1)\}$.

Comparability: Since P_i and Q_i are comparable, $Var_{O(P_1||P_2)} = Var_{OP_1} \cup Var_{OP_2} = Var_{OQ_1} \cup Var_{OQ_2} = Var_{O(Q_1||Q_2)}$. Also, it holds that

$$\begin{aligned} Var_{I(P_1||P_2)} &= (Var_{IP_1} \cup Var_{IP_2}) \setminus (Var_{OP_1} \cup Var_{OP_2}) \\ &= (Var_{IP_1} \setminus Var_{OP_2}) \cup (Var_{IP_2} \setminus Var_{OP_1}) \\ &= (Var_{IP_1} \setminus Var_{OQ_2}) \cup (Var_{IP_2} \setminus Var_{OQ_1}). \end{aligned}$$

Similarly, $Var_{I(Q_1||Q_2)} = (Var_{IQ_1} \setminus Var_{OQ_2}) \cup (Var_{IQ_2} \setminus Var_{OQ_1})$. With $Var_{IQ_i} \subseteq Var_{IP_i}$, it follows that $Var_{I(Q_1||Q_2)} \subseteq Var_{I(P_1||P_2)}$, and $P_1||P_2$ is comparable to $Q_1||Q_2$.

Equality of Inputs and Outputs: It is a prerequisite for simulation that the input and output valuations of $P_1||P_2$ and $Q_1||Q_2$ must be equal for all states in R. Formally, if $((k_1,k_2,x),(l_1,l_2,z)) \in R$, it holds that $x{\downarrow}_{O(P_i||Q_j)} = z{\downarrow}_{O(Q_i||Q_j)}$ and $x{\downarrow}_{I(Q_i||Q_j)} = z{\downarrow}_{I(Q_i||Q_j)}$. Note that $Var_{I(P_i||Q_j)} = (Var_{IP_i} \setminus Var_{OQ_j}) \cup (Var_{IQ_j} \setminus Var_{OQ_i})$. Also, it holds that $Var_{I(Q_i||Q_j)} = (Var_{IQ_i} \setminus Var_{OQ_j}) \cup (Var_{IQ_j} \setminus Var_{OQ_i})$. From $P_i||Q_j \preceq Q_i||Q_j$ it follows that for any $((k_i,l_j,y_i),(l_i,\hat{l}_j,\hat{z}_i)) \in R_i$ holds: $y_i{\downarrow}_{OP_i \cup OP_j} = \hat{z}_i{\downarrow}_{OQ_i \cup OQ_j}$ and

$$y_i{\downarrow}_{(IQ_i \setminus OQ_j) \cup (IQ_j \setminus OQ_i)} = \hat{z}_i{\downarrow}_{(IQ_i \setminus OQ_j) \cup (IQ_j \setminus OQ_i)} \ . \tag{10.7}$$

Since $y_i{\downarrow}_{P_i} = x{\downarrow}_{P_i}$, $x{\downarrow}_{OP_i} = x{\downarrow}_{OQ_i} = \hat{z}_i{\downarrow}_{OQ_i}$. Also, $\hat{z}_i{\downarrow}_{Q_i} = z{\downarrow}_{Q_i}$, so that $x{\downarrow}_{OQ_i} = z{\downarrow}_{OQ_i}$, which shows for $i = 1,2$ that the output valuations are equal. Now we show that the input valuations of $Q_1||Q_2$ are input valuations of $P_1||P_2$. Since $Var_{IQ_i} \subseteq Var_{IP_i}$, $y_i{\downarrow}_{P_i} = x{\downarrow}_{P_i}$ implies that $y_i{\downarrow}_{IQ_i} = x{\downarrow}_{IQ_i}$. It holds that $y_i{\downarrow}_{Q_j} = z{\downarrow}_{Q_j}, \hat{z}_i{\downarrow}_{Q_i} = z{\downarrow}_{Q_i}$. With (10.7) it follows that $\hat{z}_i{\downarrow}_{(IQ_i \setminus OQ_j)} = y_i{\downarrow}_{(IQ_i \setminus OQ_j)}$ and therefore it holds that $x{\downarrow}_{(IQ_i \setminus OQ_j)} = y_i{\downarrow}_{(IQ_i \setminus OQ_j)} = \hat{z}_i{\downarrow}_{(IQ_i \setminus OQ_j)} = z{\downarrow}_{(IQ_i \setminus OQ_j)}$. For $i = 1,2$ this shows that the input valuations are equal.

Simulation: Consider a state $((k_1,k_2,x),(l_1,l_2,z)) \in R$. We show that for any transition

$$(k_1,k_2,x) \xrightarrow{\alpha}_{P_1||P_2} (k_1',k_2',x') \tag{10.8}$$

there exists a transition $(l_1,l_2,z) \xrightarrow{\alpha}_{Q_1||Q_2} (l_1',l_2',z')$ with $((k_1',k_2',x'),(l_1',l_2',z')) \in R$. By the definition of parallel composition, (10.8) implies transitions

$$(k_1,x{\downarrow}_{P_1}) \xrightarrow{\alpha_1}_{P_1} (k_1',x'{\downarrow}_{P_1}) \qquad \text{and} \qquad (k_2,x{\downarrow}_{P_2}) \xrightarrow{\alpha_2}_{P_2} (k_2',x'{\downarrow}_{P_2}),$$

where $\alpha_i = \alpha$ if $\alpha \in \Sigma_i$ and $\alpha_i = \varepsilon$ otherwise for $i = 1,2$. Under the hypothesis (10.6), there exists a transition in Q_1 or in Q_2 with a corresponding label α_i. We assume $\alpha_1 = \alpha$ and a transition in Q_2 and show that there exists a transition in $[\![P_1]\!]||[\![Q_2]\!]$, which entails a transition in $[\![Q_1||Q_2]\!]$ and whose target states lie in R_1. Afterwards we must show that they lie also in R_2. An analogous argument can be made if $\alpha_2 = \alpha$ and there is a transition in Q_1.

We assume a transition $(l_2,z{\downarrow}_{Q_2}) \xrightarrow{\alpha_2}_{Q_2} (l_2',z'{\downarrow}_{Q_2})$ with

$$z'{\downarrow}_{P_1 \cap Q_2} = x'{\downarrow}_{P_1 \cap Q_2}, \tag{10.9}$$

so that the transition $(k_1,l_2,y_1) \xrightarrow{\alpha}_{[\![P_1]\!]||[\![Q_2]\!]} (k_1',l_2',y_1')$ exists with

$$y_1'{\downarrow}_{P_1} = x'{\downarrow}_{P_1} \qquad \text{and} \tag{10.10}$$

$$y_1'{\downarrow}_{Q_2} = z'{\downarrow}_{Q_2} \ . \tag{10.11}$$

Then $((k_1,l_2,y_1),(l_1,\hat{l}_2,\hat{z}_1)) \in R_1$ implies that there exist $l_1',\hat{l}_2',\hat{z}_1'$ and a transition

$$(l_1,\hat{l}_2,\hat{z}_1) \xrightarrow{\alpha}_{Q_1||Q_2} (l_1',\hat{l}_2',\hat{z}_1')$$

with $((k_1', l_2', y_1'), (l_1', \hat{l}_2', \hat{z}_1')) \in R_1$, which implies $(l_1, \hat{z}_1{\downarrow}_{Q_1}) \xrightarrow{\alpha}_{Q_1} (l_1', \hat{z}_1'{\downarrow}_{Q_1})$. We now demonstrate that the target states of the transition lie in R_2 by showing that there is a transition in $[\![P_1]\!]||[\![Q_2]\!]$, which entails a transition in $[\![Q_1||Q_2]\!]$ and whose target states lie in R_2. This creates a transition in $[\![P_2]\!]||[\![Q_1]\!]$ if $\hat{z}_1'{\downarrow}_{Q_1\cap P_2} = x'{\downarrow}_{Q_1\cap P_2}$. Note that $Var_{Q_1} \cap Var_{P_2} \subseteq Var_{IQ_1} \cup Var_{OQ_1} \subseteq Var_{P_1}$. With R_1 and (10.10) follows that $\hat{z}_1'{\downarrow}_{Q_1\cap P_2} = y_1'{\downarrow}_{Q_1\cap P_2} = x'{\downarrow}_{Q_1\cap P_2}$. Therefore the transition $(k_2, l_1, y_2) \xrightarrow{\alpha}_{[\![P_2]\!]||[\![Q_1]\!]} (k_2', l_1', y_2')$ exists with

$$y_2'{\downarrow}_{P_2} = x'{\downarrow}_{P_2} \quad \text{and} \tag{10.12}$$

$$y_2'{\downarrow}_{Q_1} = \hat{z}_1'{\downarrow}_{Q_1}\,. \tag{10.13}$$

Then $((k_2, l_1, y_2), (\hat{l}_1, l_2, \hat{z}_2)) \in R_2$ implies that there exist $\hat{l}_1', \overline{l}_2', \hat{z}_2'$ and a transition

$$(\hat{l}_1, l_2, \hat{z}_2) \xrightarrow{\alpha}_{Q_1||Q_2} (\hat{l}_1', \overline{l}_2', \hat{z}_2')$$

with $((k_2', l_1', y_2'), (\hat{l}_1', \overline{l}_2', \hat{z}_2')) \in R_2$. Note that $Var_{P_1} \cap Var_{Q_2} \subseteq Var_{IQ_2} \cup Var_{OQ_2} \subseteq Var_{P_2}$. With R_2 and (10.12) follows that $\hat{z}_1'{\downarrow}_{P_1\cap Q_2} = y_2'{\downarrow}_{P_1\cap Q_2} = x_1'{\downarrow}_{P_1\cap Q_2}$. Because (10.9) is the only restriction on $(l_2', z'{\downarrow}_{Q_2})$, we are free to chose $l_2' = \overline{l}_2'$ and $z'{\downarrow}_{Q_2} = \hat{z}_2'{\downarrow}_{Q_2}$. This fulfills the conditions of (10.5) and concludes the proof. □

Note that if P_i, Q_i are TTS-compositional HIOA, (10.3) and (10.4) in Theorem 9.16 take the simpler but equivalent form

$$[\![P_1||Q_2]\!] \preceq [\![Q_1\,||\,Q_2]\!] \quad \text{and}$$
$$[\![Q_1||P_2]\!] \preceq [\![Q_1\,||\,Q_2]\!].$$

The A/G-condition is trivially fulfilled if for every label in $Lab_{Q1} \cap Lab_{Q2}$ either Q_1 or Q_2 is non-blocking, and that in each location of $Q_1||Q_2$ either one has no invariant, so that in at least one of them timed transitions are always enabled.

The A/G-condition (10.6) looks similar to the requirement of simulation for the composed automata, but differs in two important points: Firstly, the target states are not required to lie within the relation, so there is no fixed point computation necessary. Secondly, it is only required that either one of Q_1 or Q_2 has a corresponding transition.

10.2.1 Trimming

The A/G-condition (10.6) is nearly identical to the case treated already in Part II on systems without shared variables, and a similar trimming as suggested in Sects. 7.2.1 and 7.2.2 can be applied. On has to additionally take care that $z'{\downarrow}_{P_j\cap Q_i} = x'{\downarrow}_{P_j\cap Q_i}$ holds. To operate on relations that refer to P_i, Q_1, Q_2 instead of $P_i, Q_j, Q_1||Q_2$, one can use the following relations R_i' for $(i,j) \in \{(1,2),(2,1)\}$:

$$R_i' = \{((k_i, l_j, y_i), (l_i, v_i)) | \exists \hat{l}_j, \hat{z}_i : ((k_i, l_j, y_i), (l_i, \hat{l}_j, \hat{z}_i)) \in R_i, \; v_i = \hat{z}_i{\downarrow}_{Q_i}\}. \tag{10.14}$$

Example 10.2. *Consider the level monitor and its specifications given in Ex. 10.1. Let the parameters be* $x_{over} = 200$, $x_m = 20$, $x_M = 180$, $x_l = 30$, $x_h = 176$, $\underline{r}_i = 2$, $\overline{r}_i = 5$, $\underline{r}_d = 1$, $\overline{r}_d = 3$, $\delta = 1$. *For an initial set of states* $40 \leq x \leq 160$, $d = 0$, *the verification is successful. The sets of critical states* D_{R_1} *and* D_{R_2} *are empty, and the condition of Def. 9.3 for containment of the initial states is fulfilled.*

10.2.2 Checking the Initial States

To finalize the A/G-proof, it must be shown that all the initial states of $P_1||P_2$ have a matching initial state of $Q_1||Q_2$ in R. For the simulation relation (4.10), it must be shown that for all (k_1,k_2,x), $(k_1,x{\downarrow}_{P_1}) \in Init_{P_1}$, $(k_2,x{\downarrow}_{P_2}) \in Init_{P_2}$ there exists $l_j, y_i, \hat{l}_j, \hat{z}_j$ such that:

- $((k_i,l_j,y_i),(l_i,\hat{l}_j,\hat{z}_i)) \in R_i$,
- $y_i{\downarrow}_{P_i} = x{\downarrow}_{P_i}$,
- $y_i{\downarrow}_{Q_j} = \hat{z}_j{\downarrow}_{Q_j}$, $y_i{\downarrow}_{Q_j} \in Init_{Q_j}$,
- $\hat{z}_i{\downarrow}_{Q_i} \in Init_{Q_i}$.

Recall from Sect. 4.2.3 that there exist cases in which R_1 and R_2 exist, but no simulation relation can be constructed from R_1 and R_2 that contains the initial states appropriately, even though a global R' exists and $P_1||P_2 \preceq Q_1||Q_2$ holds. A sufficient condition for the containment is that for all (k_1,l_2,y_1) with $(k_1,y_1{\downarrow}_{P_1}) \in Init_{P_1}$, $(l_2,y_1{\downarrow}_{Q_2}) \in Init_{Q_2}$ and $(l_1,v_1) \in Init_{Q_1}$ there exists $(\hat{l}_2,\hat{z}_1)$ with $\hat{z}_1{\downarrow}_{Q_1} = v_1$ such that $((k_1,l_2,y_1),(l_1,\hat{l}_2,\hat{z}_1)) \in R_1$. Alternatively a symmetric argument is valid for R_2.

10.3 Related Work

In [HMP01], Henzinger et al. present an extension of their language Masaccio and show that it supports assume/guarantee reasoning for discrete and linear hybrid automata. Their frameworks includes hierarchical modeling capabilities as well as a serial composition operator. However, the soundness of the A/G-rule is restricted to certain cases, e.g., that the components cannot deadlock internally. In our understanding this corresponds to the case of unrestricted inputs and input-enabledness, and is contained as a special case in our A/G-conditions. The example provided in this paper relies on periodic sampling of the states, which fulfills the sufficient condition for compositionality given by Theorem 9.14.

Chapter 11

PHAVer - A Novel Verification Tool for Hybrid Systems

Systems with discrete as well as continuous dynamics, i.e., hybrid systems, are notoriously complex to analyze, and the algorithmic verification of hybrid systems remains a challenging problem, both from a theoretic point of view as well as from the implementation side. Ideally, one would like to obtain either an exact result or a conservative overapproximation of the behavior of the system, e.g., as the set of reachable states. An exact computation is possible with linear hybrid automata (LHA) [Hen96], which are defined by linear predicates and piecewise constant bounds on the derivatives. They were proposed and studied in detail by Henzinger et al. (see, e.g., [Ho95] for an extensive discussion) who presented in 1995 a tool called HyTech that could perform various computations with such systems [HHWT97]. It featured a powerful input language and functionality, but suffered from a major flaw: its exact arithmetic was using limited digits, thus quickly leading to overflow errors. While it was successfully used to analyze a number examples, see, e.g., [HWT96b, Tom96, KSF+99, HPWT01, CEG+01], the overflow problem prevents any application to larger systems.

The valuable experiences with HyTech have prompted a number of suggestions for improvement, a summary of which can be found in [HPWT01]. We address the most pressing ones with PHAVer (Polyhedral Hybrid Automaton Verifyer), a new tool for analyzing hybrid automata with the following features:

- exact and robust arithmetic with unlimited precision,
- on-the-fly overapproximation of piecewise affine dynamics,
- improved algorithms and termination heuristics,
- support for compositional and assume-guarantee reasoning. [1]

[1] Not addressed are more advanced input capabilities like hierarchy, templates and directional communication labels, since we consider these easily and more appropriately handled by a GUI-frontend or editor. A simple procedure for modeling directional communication with hybrid automata can be found in [HLFE02].

PHAVer's extended functionality and computational robustness open up new application domains as well as research issues that were abandoned because of the limitations of previous implementations. Exact arithmetic brings, in addition to the satisfaction and beauty of formal correctness, the significant advantage of a separation of concerns. Problems of convergence, combinatorial explosion and nondeterminism can be identified as such, which is very difficult if they are intertwined with numerical difficulties. In our experience, this greatly aids in the understanding of systems and analysis methods, since without exactness one can be quickly misled to attribute complications to numerics. We give a brief overview of the functionality of PHAVer, and illustrate our on-the-fly algorithm for overapproximating piecewise affine dynamics with LHA, which partitions locations with user-specified constraints according to a prioritization function. The applicability and competitiveness of PHAVer is demonstrated with a navigation benchmark proposed in [FI04].

Computations in PHAVer use convex polyhedra as the basic data structure, and apply the *Parma Polyhedra Library* (PPL) by Roberto Bagnara et al. [BRZH02]. The PPL supports for closed and non-closed convex polyhedra and exact arithmetic with unlimited digits. While floating point computations are inherently faster than exact arithmetic, there is a considerable overhead and tweaking involved, e.g., to solve problems like containment and equality, that is unnecessary when computing exact. On the other hand, exact computations usually lead to a significant if not exponential rise in the length of the coefficients of polyhedra. To manage the computational complexity, we propose overapproximating methods to limit the number of bits in the coefficients and to limit the number of constraints used to describe polyhedra. Experimental results show that the overapproximation is negligible, and the speedup amounts to more than an order of magnitude for systems that are still computable with exactness. For more complex systems, where the analysis with limited complexity still performs well, exact methods very rapidly reach the computational limits.

In the following section, we present our experimental implementation of simulation and assume/guarantee reasoning in PHAVer and illustrate the PHAVer syntax with some small examples. In Sect. 11.2 we give an overview on the reachability algorithm implemented in PHAVer, and the operators involved. We describe the on-the-fly overapproximation of affine dynamics, and give experimental results for a navigation benchmark. In Sect. 11.3, we propose methods to reduce the complexity of polyhedral computations by conservative overapproximation. Experimental results for a tunnel-diode oscillator circuit show the effectiveness of the approach. The chapter concludes with Sect. 11.4 on related work.

11.1 Simulation Checking and Assume/Guarantee-Reasoning

We illustrate the simulation checking in PHAVer with the tank level monitor example from Chapter 8. It is a paradigmatic example for proving invariance of a system with feedback controller with assume/guarantee-reasoning. Under the assumption that the controller guarantees the invariant, the controlled system remains in a restricted operating region that corresponds to the invariant. The assume/guarantee conditions break this circularity, which otherwise would be unsound. We first present a simulation check for the composed system, and then verify the same property with decomposed specifications and assume/guarantee reasoning. An extended version of the tank level monitor example will be used to present experimental results for the computational gains.

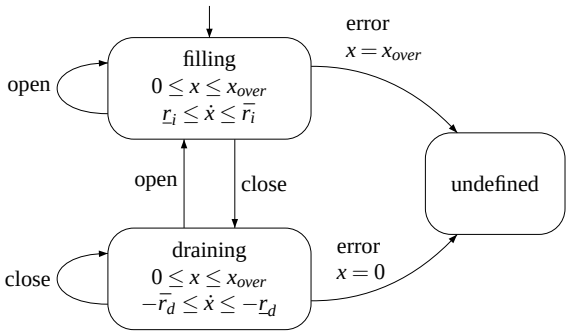
The system consists of two components, the tank P_1 and the controller P_2, which we model with linear hybrid automata. The hybrid I/O-automaton modeling the tank is shown in Fig. 11.1(a), and the corresponding PHAVer model in Fig. 11.1(b). The tank has an inlet valve that can be opened and closed by the controller, and a constant outflow that results in a net outflow if the inlet valve is closed, and a net inflow if it is open. The model boundaries are marked with transitions with a label *error* in order to detect when the model runs the danger of surpassing them.

We give a brief summary of the syntax. The definition of the automaton begins with the declaration of the controlled variable x and the synchronization labels. Then the location filling is defined with its invariant, and with its time derivative given as a linear predicate over $\dot{x}$. Note that in the declaration of the derivative `wait {...}`, the derivative is simply written as `x`. The transitions are defined in the form

`when` *guard* `sync` *label* `do {`*transition relation*`} goto` *target location*`;`

The transition relation and guard are redundant predicates are somewhat redundant, but guards are often used in modeling hybrid systems and therefore convenient to include. The guards are combined with the transition relation by the parser. The transition relation is defined as a linear predicate over `x` and `x'`, where `x'` denotes the value of `x` after the transition. Unrestricted predicates that are denoted by `True`. The initial states are declared at the end with a comma separated list of locations names and predicates adjoined to each location by `&`.

A HIOA model of the controller and its PHAVer implementation are shown in Fig. 11.2. It has a timer d, declared as a controlled variable, and measures the tank level x, which is declared as an input variable. The controller samples the tank level every δ seconds and decides instantly whether to open the valve, close it, or do nothing. It is the task of the controller to keep the tank level x within certain limits, i.e., to ensure invariance of an interval $[x_m, x_M]$ for the variable x. This is a global specification for the composed system, and is modeled by the HIOA Q shown in Fig. 11.3. The invariant interval is specified as in the invariant of Q, and self-loops permit all labels except *error*. The specification is therefore only fulfilled if the system remains within the

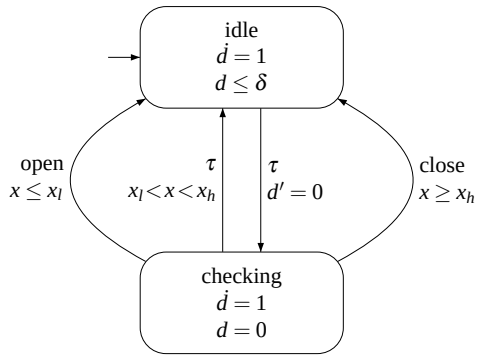


(a) Hybrid automaton

```
automaton P1
state_var:   x;
synclabs:       open,close,error;
loc filling: while 0 <= x & x <= x_over wait {r_il <= x & x <= r_ih};
  when True       sync close do {x'==x} goto draining;
  when True       sync open  do {x'==x} goto filling;
  when x==x_over  sync error do {x'==x} goto undefined;
loc draining: while 0 <= x & x <= x_over wait {-r_dh <= x & x <= -r_dl};
  when True       sync open  do {x'==x} goto filling;
  when True       sync close do {x'==x} goto draining;
  when x==0       sync error do {x'==x} goto undefined;
loc undefined: while True wait {True};
initially: filling & x_0l <= x & x <= x_0h;
end
```

(b) Input file

Figure 11.1: Model of tank P_1

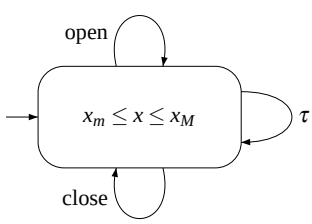


(a) Hybrid automaton

```
automaton P2
state_var:   d;
input_var:   x;
synclabs:       open,close,tau;
loc idle: while 0 <= d & d <= delta wait {d==1};
  when True sync tau do {d'==0} goto checking;
loc checking: while d == 0 wait {d==1};
  when x <= x_l sync open   do {d'==d} goto idle;
  when x >= x_h sync close  do {d'==d} goto idle;
  when x_l < x & x < x_h sync tau do {d'==d} goto idle;
initially: idle & d==0;
end
```

(b) Input file

Figure 11.2: Model of controller P_2



(a) Hybrid automaton

```
automaton Q
state_var: x;
synclabs:  open,close,error,tau;
loc always:
  while x_m <= x & x <= x_M
    wait {True};
  when True sync open   do {True} goto always;
  when True sync close  do {True} goto always;
  when True sync tau    do {True} goto always;
initially:
  always & x_m <= x & x <= x_M;
end
```

(b) Input file

Figure 11.3: Model of specification Q

model boundaries, which were staked off by error transitions.

We will first verify the composed system by checking for simulation, and then check for assume/guarantee simulation. We define coefficients

```
x_over := 200;   x_m   := 20;   x_M   := 180;
x_l    := 30;    x_h   := 175;  r_il  := 2;
r_ih    := 5;    r_dl  := 1;    r_dh    := 3;
delta   := 1;    x_0l  := 40;   x_0h    := 160;
```

PHAVer by default initializes the simulation relation with the convex hull of the reachable set. We turn this option off with `SIM_PRIME_WITH_REACH=false;` Then we define the system with `sys=P1&P2;` and compute the simulation relation R with the command `R=get_sim(sys,Q);` This produces the following output:

```
        Composing automata P1 and P2
          ......

  Checking P1~P2 <= Q
  -------------------
        Assigning discretely reachable states to simulation relation
      Fixpoint computation on simulation relation
          .....
        Simulation Relation converged after 5 iterations.
        State relation size:4 loc pairs, 20 conv. polyh.
    Ini states in simulation relation: yes
```

First, P_1 and P_2 are composed. Then the simulation relation is initialize with unconstrained predicates for the locations that are reachable by looking only at which locations are connected by discrete transitions. Then the simulation relation is computed with a standard fixpoint algorithm similar to the one presented in Sect. 6.3. Finally, the initial states are tested for containment. We can out put the relation with `R.print;` and the output is shown in Fig. 11.4. For each location pair, a disjunction of convex linear predicates is shown. The conjunctions are represented by commata, and one element of the disjunction per line is shown. We now demonstrate the assume/guarantee verification of the specification. First, decomposed specifications Q_i are chosen for each component of the system. For the A/G-specification Q_2 of the controller we chose the invariant set, and it is equal to the global specification Q except that x is an input instead of a controlled variable. The A/G-specification Q_1 of the plant, i.e., the component to be controlled, is its behavior restricted to this invariant set. Both are shown in Fig. 11.5.

First we verify that the decomposed specifications indeed guarantees the global specification. With `spec=Q1&Q2;` and `is_sim(spec,Q);` we get

```
        Composing automata Q1 and Q2
          ..

  Checking Q1~Q2 <= Q
  -------------------
```

```
State relation:
[0,0]:
1(3): x - x' = 0, x > 200
2(3): x - x' = 0, -x' >= -200, d > 1
3(3): x - x' = 0, -d > 0, -x' >= -200
4(3): x - x' = 0, -x > 0, -d >= -1, d >= 0
5(3): x - x' = 0, -x + 5*d >= -175, -d >= -1, d >= 0, x >= 20
[2,0]:
1(3): x - x' = 0, d > 0
2(3): x - x' = 0, -d > 0
3(3): d = 0, x - x' = 0, x > 200
4(3): d = 0, x - x' = 0, -x >= -180, x >= 20
5(3): d = 0, x - x' = 0, -x > 0
[3,0]:
1(3): x - x' = 0, -x > 0
2(3): x - x' = 0, -x' >= -200, x' >= 0, d > 1
3(3): x - x' = 0, -d > 0, -x' >= -200, x' >= 0
4(3): x - x' = 0, x > 200
5(3): x - x' = 0, -x >= -180, -d >= -1, d >= 0, x + 3*d >= 23
[5,0]:
1(3): x - x' = 0, d > 0
2(3): x - x' = 0, -d > 0
3(3): d = 0, x - x' = 0, -x > 0
4(3): d = 0, x - x' = 0, x > 200
5(3): d = 0, x - x' = 0, -x >= -180, x >= 20
State relation size:
4 loc pairs, 20 conv. polyh.
```

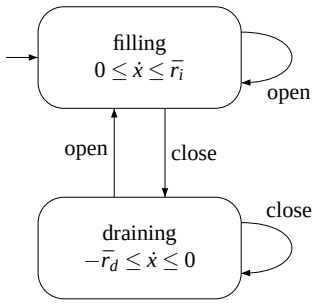
Figure 11.4: Simulation relation for $P \preceq Q$

```
    Priming simulation relation (convex hull reach)
    Computing reachable states of Q1~Q2
      ...
    Computing reachable states of Q
      .
  Fixpoint computation on simulation relation
      ..
    Simulation Relation converged after 2 iterations.
    Found no bad states.
    State relation size:2 loc pairs, 2 conv. polyh.
Ini states in simulation relation: yes
```

We recall the structure of the assume/guarantee proof, and that for linear hybrid automata we need not differentiate between the TTS-semantics and the automata themselves:

$$\frac{\begin{array}{c} P_1 \,||\, Q_2 \;\preceq\; Q_1 \,||\, Q_2 \\ Q_1 \,||\, P_2 \;\preceq\; Q_1 \,||\, Q_2 \\ \textit{A/G conditions} \end{array}}{P_1 \,||\, P_2 \;\preceq\; Q_1 \,||\, Q_2}.$$

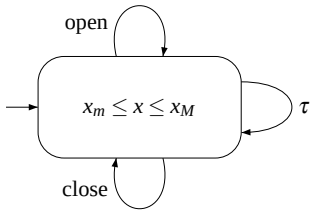
The reasoning is obviously circular: With the first inequality, we check whether the plant P_1 restricted to the invariant set Q_2 indeed exhibits the restricted dynamics Q_1. In



(a) Hybrid automaton Q_1

```
automaton Q1
state_var: x;
synclabs:  open,close,error;
loc filling:
  while 0 <= x & x <= x_over
    wait {0 <= x & x <= r_ih};
  when True sync close do {x'==x} goto draining;
  when True sync open  do {x'==x} goto filling;
loc draining:
  while 0 <= x & x <= x_over
    wait {-r_dh <= x & x <= 0};
  when True sync open  do {x'==x} goto filling;
  when True sync close do {x'==x} goto draining;
initially:
  filling & x_0l <= x & x <= x_0h;
end
```

(b) Input file for Q_1



(c) Hybrid automaton Q_2

```
automaton Q2
input_var:   x;
loc always:
   while x_m <= x & x <= x_M
   wait {True};
   when
initially:
   always & x_m <= x & x <= x_M;
end
```

(d) Input file for Q_2

Figure 11.5: A/G-specifications

the second inequality, we verify that the controller P_2, given the restricted plant dynamics, guarantees the invariant. The assume-guarantee conditions will ensure that this circularity is broken, i.e., that both remain always in the region of their mutual guarantees. The assume/guarantee verification is implemented as a hybrid version of the composite trimming algorithm in Fig. 4.5 of Sect. 4.2.2, see Sect. 7.2.2 for the hybrid formulation. We start the verification with the command `agc_sim(P1,P2,Q1,Q2);` The simulation relations R_1 and R_2 for both inequalities are computed. First the one for $P_1 \,||\, Q_2 \preceq Q_1 \,||\, Q_2$:

```
Checking A/G-simulation P1&P2<=Q1&Q2
------------------------------------
      Composing automata Q1 and Q2
        ..
      Getting simrel R1
      Composing automata P1 and Q2
        ...
      Priming simulation relation (convex hull reach)
      Computing reachable states of P1~Q2
        ...
      Computing reachable states of Q1~Q2
        ...
    Fixpoint computation on simulation relation
        ..
      Simulation Relation converged after 2 iterations.
      Found no bad states.
      State relation size:2 loc pairs, 2 conv. polyh.
```

Then the one for $P_2 \,||\, Q_1 \preceq Q_1 \,||\, Q_2$:

```
      Composing automata P2 and Q1
        ....
      Priming simulation relation (convex hull reach)
      Computing reachable states of P2~Q1
        ...............................................
        ...............................................
        ........
      Computing reachable states of Q1~Q2
        ...
    Fixpoint computation on simulation relation
        ....
      Simulation Relation converged after 4 iterations.
      Found no bad states.
      State relation size:4 loc pairs, 4 conv. polyh.
```

Then the sets of potentially violating states, D^{tr} for discrete transitions and D^{te} for time elapse, are computed and subtracted from R_1 and R_2, which must afterwards again be subjected to a fixpoint computation to turn them back into simulation relations. The check concludes with testing whether the initial states are contained:

```
        Getting Dtr and Dte
        .....
```

```
      .....
  Fixpoint computation on simulation relation
      ..
    Simulation Relation converged after 2 iterations.
    Found no bad states.
    State relation size:2 loc pairs, 2 conv. polyh.
  Fixpoint computation on simulation relation
      ....
    Simulation Relation converged after 4 iterations.
    Found no bad states.
    State relation size:4 loc pairs, 4 conv. polyh.
    Ini states in R1: yes
    Ini states in R2: yes
Ini states in simulation relation: yes
```

The A/G-check came to the same result, albeit not with an increase in speed due to the small size difference between the system and the A/G-specifications. The simulation check of the composed system took 1.9 s, while the A/G-verification took 6.2 s in total.

The tank level monitor in the above form is not large enough for the assume/guarantee reasoning to pay off. We extended both models with additional locations to obtain a parameterized result over the size of the model. The controller was extended by n_C locations and a min. and max. sampling time. The locations are entered after the control decision, representing other tasks that might take up the controller's time. The simple tank model was replaced by a LHA-approximation of a nonlinear model, which was presented as Example 5.3. The square-root characteristic of the outflow of the tank was approximated by n_T intervals, resulting in $2n_T$ locations, one each for the draining and filling modes.

Table 11.1 show the results for an Intel Pentium 4M with 1.9GHz, 768MB RAM. With increasing $n = n_T = n_C$ the A/G-reasoning (A/G-Sim.) shows a clear advantage over simulation checking of the composed system (Sim.), and even over a convex-hull reachability analysis (Reach.). This correlates with the size of the simulation relations, $|R|$ for the composed analysis and $\sum |R_i| = |R_1| + |R_2|$ for A/G-reasoning, each measured in the number of locations. For both analyses with the composed system, i.e., simulation and reachability, the composition cost becomes the dominating factor. However, even the net time of the reachability analysis lies for $n = 80$ at over 60 s, which clearly demonstrates the superiority of the assume/guarantee approach.

11.2 Reachability Analysis

A reachability analysis computes all states that are connected to the initial states by a run. PHAVer enhances the standard fixpoint computation algorithm for reachability with operators for the partitioning of locations and the simplification of sets of states. The partitioning of locations is used when affine dynamics are overapproximated with

Table 11.1: Analysis of extended tank level monitor model

n	Sim.	Reach.	A/G-Sim.	$\lvert R\rvert$	$\sum \lvert R_i\rvert$
1	0.46 s	0.30 s	1.21 s	4	6
10	10.67 s	3.67 s	5.28 s	183	42
20	33.70 s	14.49 s	9.76 s	490	82
40	197.53 s	109.07 s	19.44 s	2030	162
80	1826.59 s	1217.35 s	43.13 s	9312	318

LHA-dynamics, where locations are split into smaller parts to improve the accuracy. The simplification operator fulfills two purposes: Firstly, the overapproximation of sets of states with a simpler representation keeps the complexity from growing beyond computationally manageable limits. We propose methods to limit the number of bits and the number of constraints used in describing sets of states. Secondly, since termination is not guaranteed for linear hybrid automata, overapproximation of the sets of states as well as the set of derivatives can be used to accelerate convergence and possibly force termination by reducing the model to a class where reachability is decidable. The challenge lies in trading speed, termination and resource consumption against the loss of accuracy.

The algorithm used in PHAVer for computing the set of reachable states is shown in Fig. 11.6. We give a brief summary of the operators used. Let X, Y and $Y_1, \ldots, Y_z$ be arbitrary sets of states, each described by a set of convex polyhedra for each location.

Post-Operators: The operator *time_elapse*(X,Y) computes the successors of a set of states X by letting time elapse according to a set Y that attributes a set of derivatives to each location. The successors of discrete transitions are given by *trans_post*(X). A detailed description can be found in [Ho95].

Overapproximating Operators: The operator *cheap_difference*(X,Y) computes a overapproximation of $X \setminus Y$ by returning the polyhedra in Y that are not individually contained in some polyhedra of X. The gain in speed usually far outweighs the fact that more states are iterated than necessary [Ho95]. With *union_approx*(X,Y), the union of new states X and old states Y can be overapproximated, e.g., by using the convex hull. This must take place before the *time_elapse* operator in order to be sound. For exact computation, the operator is skipped. If there are no new states for a location then the operator returns the empty set for that location. The *simplify* operator is used to reduce the complexity the representation of states by overapproximation. It can also be applied to the set of derivatives in the location. Current options in PHAVer for *simplify* include a bounding box overapproximation, limiting the number of bits used by the coefficients of constraints, and limiting the number of constraints.

procedure *GetReach*
  Input: a set of initial states S_I
  Output: the set of states S_R reachable from S_I
  $(S_I, \{S_I\}) := partition_loc(S_I, \{S_I\})$**;**
  $W, S_R := time_elapse(S_I)$**;**
  while $W \neq \emptyset$ **do**
    $N := trans_post(W)$**;**
    $(N, (S_I, S_R, W)) := partition_loc(N, (S_I, S_R, W))$**;**
    $N := cheap_difference(N, S_R)$**;**
    $N := union_approx(N, S_R)$**;**
    $N := simplify(N)$**;**
    $N := time_post(N, simplify(time_deriv(N, Inv)))$**;**
    $S_R := S_R \cup N$**;**
    $W := N$
  od.

Figure 11.6: Reachability Algorithm in PHAVer

Partitioning Operators: The operator $partition_loc(X, (Y_1, \ldots, Y_z))$ partitions the locations with states in X as described in Sect. 11.2.1 and maps the states in $Y_1, \ldots, Y_z$ to the new set of locations. With $time_deriv(X, Y)$ the set of derivatives is computed that any state in X might exhibit, provided that the states are confined to Y:

$$time_deriv(X,Y) = \{(l, \dot{f}(t)) | \exists (l,v) \in X, f \in Act(l), t \in \mathbb{R}^{\geq 0} : (f(0) = v \wedge \forall t', 0 \leq t' \leq t : f(t') \in Y)\} \quad (11.1)$$

In the following two sections we propose methods for limiting the number of bits and constraints in polyhedral computations. The partitioning of locations and the over-approximation of affine dynamics with LHA dynamics are discussed in Sect. 11.2.1.

11.2.1 On-the-fly Overapproximation of Affine Dynamics

While PHAVer's computations are based on linear hybrid automata models, it also accepts affine dynamics, which are then overapproximated conservatively. The approximation error depends on the size of the location and the dynamics, so PHAVer offers to partition reachable locations during the analysis. The partitioning takes place by splitting locations recursively along user-defined hyperplanes until a minimum size is reached or the dynamics are sufficiently partitioned.

The ***relaxed affine dynamics*** are given by a convex linear predicate for its derivatives, i.e., a conjunction of constraints

$$a_i^T \dot{x} + \hat{a}_i^T x \bowtie_i b_i, \qquad a_i, \hat{a}_i \in \mathbb{Z}^n, b_i \in \mathbb{Z}, \bowtie_i \in \{<, \leq, =\},\ i = 1, \ldots, m. \quad (11.2)$$

for each location. In the following, we assume the equalities to be modeled using conjuncts of pairs of inequalities. In a location loc, the constraints (11.2) are overapproximated conservatively with constraints of the form $\alpha_i \dot{x} \bowtie_i \beta_i$, $\alpha_i \in \mathbb{Z}^n, \beta_i \in \mathbb{Z}$, by finding the infimum of (11.2) inside the invariant $Inv(loc)$. Let

$$p/q = \inf_{x \in Inv(loc)} \hat{a}_i^T x, \qquad p, q \in \mathbb{Z}.$$

If p/q exists, the set of $\dot{x}$ that fulfill (11.2) is bounded by $a_i^T \dot{x} \bowtie_i b_i - p/q$, otherwise the constraint must be dropped. The linear constraint on $\dot{x}$ is then given by $\alpha_i = qa_i$, $\beta_i = qb_i - p$.

The resulting overapproximation error depends on the size of the locations and the dynamics but can be made arbitrarily small by defining suitably small locations. PHAVer does so by recursively splitting a location along a suitable hyperplane chosen from a user-provided set. The splitting is repeated in reachable locations until a certain threshold, e.g., a minimum size, is reached. We account for the dynamics of the system using the spatial angle that is spanned by the derivatives in a location. Let the ***spread*** $\sphericalangle(X)$ of a set of valuations be defined as

$$\sphericalangle(X) = \arccos \min_{x,y \in X} x^T y / |x||y|\}$$

and the spread $\sphericalangle_{deriv}(loc, X, Y)$ of the derivatives of states X confined to states Y in location loc as

$$\sphericalangle_{deriv}(loc, X, Y) = \sphericalangle\left(\{v | (loc, v) \in time_deriv(X, Y)\}\right).$$

The spread of the derivatives is used in two ways: The partitioning of a location is stopped once the spread is smaller than a given minimum, or the constraints are prioritized according to the spread of the derivatives in the location after the splitting.

Recall that a hyperplane h is defined by an equation $a_h^T x = b_h$, where the normal vector a_h determines its direction and the inhomogeneous term b_h its position. Let the ***slack*** of h in a location loc be defined by

$$\Delta(a_h) = \max_{x \in Inv(loc)} a_h^T x - \min_{x \in Inv(loc)} a_h^T x.$$

In PHAVer, the user provides a list of candidate normal vectors $a_{h,i}$ and the minimum and maximum slack that the hyperplanes will have in the partitioned locations, i.e.,

$$Cand = \{(a_{h,1}, \Delta_{min,1}, \Delta_{max,1}), \ldots, (a_{h,m}, \Delta_{min,m}, \Delta_{max,m})\}.$$

This allows the user to include expert knowledge by choosing planes and location sizes suitable for the system. The candidate hyperplanes are prioritized according to a user-controlled list of criteria. We consider the criteria to be a map

$$split_crit : \{a^T x \bowtie b | a \in \mathbb{Z}^n, b \in \mathbb{Z}\} \times Loc \times 2^{S_H} \mapsto (\mathbb{R} \cup \infty \cup -\infty)^z$$

that attributes a z-tuple of prioritizing measures, evaluated lexicographically, to each constraint, and takes into account set of valuations considered of interested. Two special symbols are included: ∞ voids the constraint, but it can be overruled by $-\infty$, which takes precedence over all other factors. The currently implemented measure $split_crit(a^T x \bowtie b, loc, N)$, where N is the set of reachable states in the location, takes into account the following:

1. Prioritize constraints according to their slack:

$$split_crit_1 = \begin{cases} \Delta(a_h)/\Delta_{min,h} & \text{if} \Delta(a_h) > \Delta_{min,h}, \\ \infty & \text{otherwise.} \end{cases}$$

2. Prioritize constraints that have reachable states only on one side:

$$split_crit_2 = \begin{cases} 1 & \text{if } \exists x, x' \in N : a^T x < b \wedge a^T x' > b \\ 0 & \text{otherwise.} \end{cases}$$

3. Prioritize constraints according to the spread of the derivatives. Discard constraint if a minimum spread $\lhd_{min}$ is reached and the slack is smaller than $\Delta_{max,h}$:

$$split_crit_3 = \begin{cases} -\lhd_{deriv}(loc, N, Inv) & \text{if } \lhd_{deriv}(loc, N, Inv) \geq \lhd_{min} \\ & \qquad \vee\ \Delta(a_h) > \Delta_{max,h}, \\ \infty & \text{otherwise.} \end{cases}$$

4. Prioritize constraints according to the derivative spread after the constraint is applied:

$$split_crit_4 = -\max\{\lhd_{deriv}(loc, N, \{(l,x) \in Inv \mid a^T x \leq b\}), \\ \lhd_{deriv}(loc, N, \{(l,x) \in Inv \mid a^T x \geq b\})\}.$$

For efficiency, the partitioning is applied on-the-fly as shown in the reachability algorithm of Fig. 11.6. The algorithms for splitting a location, and partitioning the location with the prioritized candidate constraints are shown in Fig. 11.7 and Fig. 11.8.

procedure *SplitLocation*

Input: Hybrid I/O-automaton $H = (Loc, Var_S, Var_I, Var_O, Lab, \rightarrow, Act, Inv, Init)$,
location loc, constraint $a_i^T x \bowtie_i b_i$, splitting label τ_H,
list $\{Y_1, \ldots, Y_n\}$ of set of states of H for remapping

Output: Hybrid I/O-automaton H with split location loc

Loc**:=**$\{l \in Loc \mid l \neq loc\} \cup \{(loc, \leq), (loc, \geq)\}$**;**

$\rightarrow$**:=**$\{(l, a, \mu, l') \in \rightarrow \mid l \neq loc \wedge l' \neq loc\}$
$\cup \{(l, a, \mu, (loc, \leq)), (l, a, \mu, (loc, \geq)) \mid (l, a, \mu, loc) \in \rightarrow\}$
$\cup \{((loc, \leq), a, \mu, l'), ((loc, \geq), a, \mu, l') \mid (loc, a, \mu, l) \in \rightarrow\}$
$\cup \{(l, \tau_H, \{x' = x | x \in Var\}, l') \mid l, l' \in \{(loc, \leq), (loc, \geq)\}\}$**;**

Act**:=**$\{l \mapsto x(t) \in Act \mid l \neq loc\}$
$\cup \{(loc, \bowtie) \mapsto x(t) \mid loc \mapsto x(t) \in Act, \bowtie \in \{\leq, \geq\}\}$**;**

for $S \in \{Y_1, \ldots, Y_n\} \cup \{Inv, Init\}$ **do**
S**:=**$\{(l, x) \in S \mid l \neq loc\}$
$\cup \{((loc, \bowtie), x) \mid (loc, x) \in S \wedge a_i^T x \bowtie b_i, \bowtie \in \{\leq, \geq\}\}$
od.

Figure 11.7: Splitting a location along a hyperplane

procedure *partition_loc*

Input: Hybrid I/O-automaton $H = (Loc, Var_S, Var_I, Var_O, Lab, \rightarrow, Act, Inv, Init)$,
set of investigated states N, set of candidate constraints
$Cand = \{(a_{h,1}, \Delta_{min,1}, \Delta_{max,1}), \ldots, (a_{h,m}, \Delta_{min,m}, \Delta_{max,m})\}$,
list $\{Y_1, \ldots, Y_n\}$ of set of states of H for remapping

Output: Hybrid I/O-automaton H with locations in N partitioned

for $loc \in \{l \in Loc | \exists x : (l, x) \in N\}$ **do**
do
for $i = 1, \ldots, m$ **do**
b_i**:=**$1/2 \left(\max_{x \in Inv(loc)} a_{h,i}^T x + \min_{x \in Inv(loc)} a_{h,i}^T x \right)$**;**
c_i**:=**$split_crit(a_{h,i}^T x = b_i, loc, N)$
od;
k**:=**$\operatorname{argmin}_{i=1,\ldots,m} c_i$**;**
if $\infty \notin c_k \vee -\infty \in c_k$ **then**
$SplitLocation(H, loc, a_{h,k}^T x = b_k, \tau_H, \{Y_1, \ldots, Y_n\})$
od
while k **exists and** $\infty \notin c_k \vee -\infty \in c_k$ **od**
od.

Figure 11.8: Partitioning states with a set of candidate constraints

11.2.2 Example: Navigation Benchmark

We illustrate the reachability analysis of PHAVer with a benchmark proposed in [FI04]. It models an object moving in a plane, and following dynamically a set of desired velocities $v_d(i) = (sin(i\pi/4), cos(i\pi/4))^T$, $i = 0,\ldots,7$, where i is attributed to each unit square in the plane by a given map M. A special symbol `A` denotes the set of target states, and `B` denotes the set of forbidden states for the object. We verified that the forbidden states are not reachable for the instances shown in Fig. 11.9, whose maps are given by:

$$M_{\text{NAV01}} = M_{\text{NAV02}} = M_{\text{NAV03}} = \begin{pmatrix} \mathtt{B} & 2 & 4 \\ 2 & 3 & 4 \\ 2 & 2 & \mathtt{A} \end{pmatrix}, M_{\text{NAV04}} = \begin{pmatrix} \mathtt{B} & 2 & 4 \\ 2 & 2 & 4 \\ 1 & 1 & \mathtt{A} \end{pmatrix}.$$

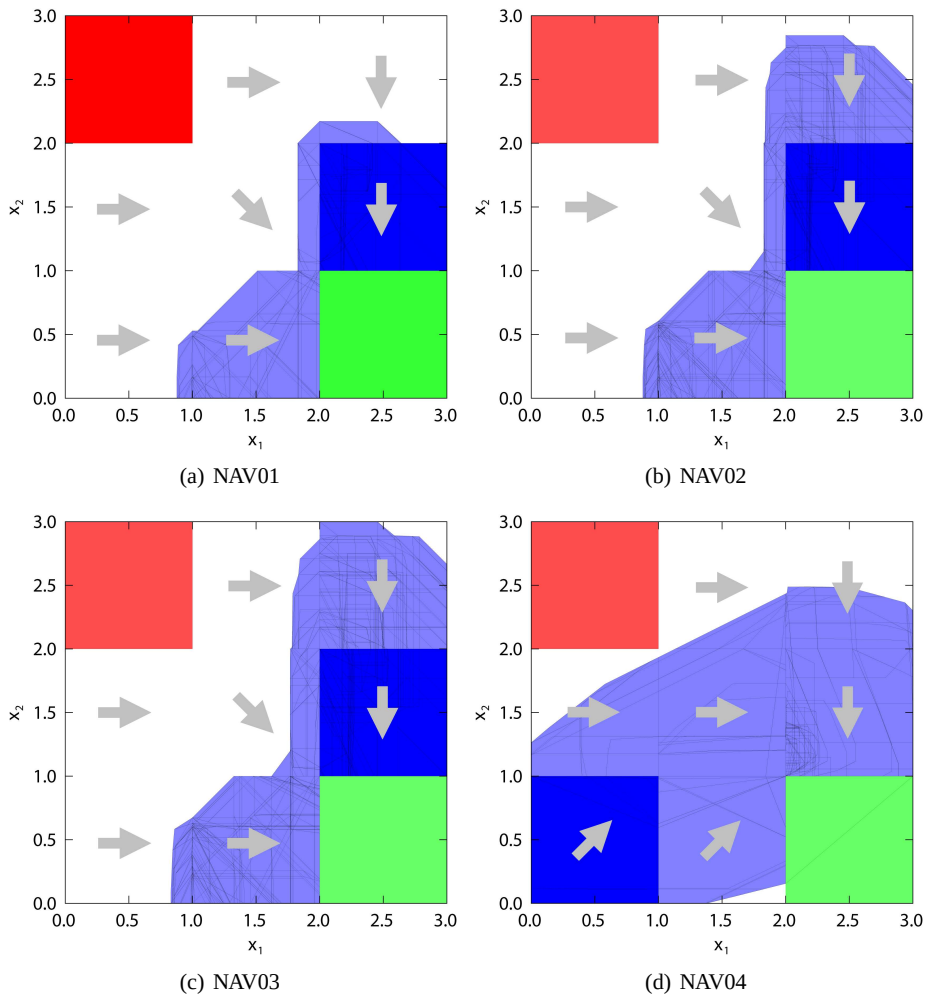
The dynamics of the 4-dimensional state vector $(x_1, x_2, v_1, v_2)^T$ are given by

$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} 0 & I \\ 0 & A \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} - \begin{pmatrix} 0 \\ A \end{pmatrix} \begin{pmatrix} 0 \\ v_d(i) \end{pmatrix}, \text{ with } A = \begin{pmatrix} -1.2 & 0.1 \\ 0.1 & -1.2 \end{pmatrix}.$$

The initial states for for NAV01–NAV03 are defined by $x_0 \in [2,3] \times [1,2]$, for NAV04 by $x_0 \in [0,1] \times [0,1]$, and

$$\begin{array}{ll} v_{0,\text{NAV01}} \in [-0.3, 0.3] \times [-0.3, 0], & v_{0,\text{NAV02}} \in [-0.3, 0.3] \times [-0.3, 0.3], \\ v_{0,\text{NAV03}} \in [-0.4, 0.4] \times [-0.4, 0.4], & v_{0,\text{NAV04}} \in [0.1, 0.5] \times [0.05, 0.25]. \end{array}$$

As splitting constraints we use $Cand = \{(v_1, \delta_1, \infty), (v_2, \delta_2, \infty)\}$, where appropriate δ_i were established by some trial-and-error runs, and $(split_crit_1)$ as splitting criterion. Note that x_1, x_2 need not be partitioned, since they depend only on v. The other analysis parameters were left at their default setting. While we need to specify bounds for the analysis region, we can handle the unbounded case by checking that the reachable state space is strictly contained in the analysis region. All instances shown were obtained with a-priori bounds of $[-2,2]$ on the velocities, and the reachable velocities remained within an interval $[-1.1, 1.1]$, which confirms our a-priori bounds as valid. Figure 5.2 shows the set of reachable states computed by PHAVer as a result. Computation times and memory consumption are shown in Table 11.2, and were obtained on a Pentium IV, 1.9GHz with 768 MB RAM running Linux. For the instances NAV01–NAV03, the analysis was fairly straightforward, with $\delta_i = 0.5$. For the instance NAV04 we had to set $\delta_i = 0.25$, and the analysis did not terminate at first. We applied a heuristic: The convex hull was computed for the first 20 iterations for speed, then switched to normal reachability, and at iteration 40 a bounding box simplification was triggered manually. In comparison, for a predicate abstraction tool the following times were reported in [Iva03]: For NAV01–NAV03 34s, 153s (68MB) and 152s (180MB), respectively, on a Sun Enterprise 3000 (4 x 250 MHz UltraSPARC) with 1 GB RAM.



(a) NAV01 (b) NAV02

(c) NAV03 (d) NAV04

Figure 11.9: Reachable states of the navigation benchmark projected to the x_1, x_2-plane (initial states darkest, arrows show terminal velocity for location)

Table 11.2: Experimental results for the navigation benchmark

Instance	Time	Memory	Iter.	Automaton Loc.	Automaton Trans.	Reachable Set Loc.	Reachable Set Polyh.
NAV01	34.73 s	62.6 MB	13	141	3452	79	646
NAV02	62.16 s	89.7 MB	13	153	3716	84	1406
NAV02[i]	41.05 s	53.7 MB	13	148	3661	84	84
NAV03	61.88 s	90.0 MB	13	153	3716	84	1406
NAV04[ii]	225.08 s	116.3 MB	45	267	7773	167	362

[i] convex hull, [ii] convex hull up to iter. 20, bounding box at iter. 40

11.3 Managing Complexity

A set of symbolic states is described by a linear predicate, the convex sub-predicates of which define convex polyhedra, which in turn are described by a set of constraints. In exact fixpoint computations with polyhedra, the size of numbers in the predicate as well as the number of constraints typically increases unless the structure of the hybrid system imposes boundaries, e.g., with resets or invariants. To keep the complexity manageable, we propose the simplification of complex polyhedra in a strictly conservative fashion by limiting the number of bits, i.e., the size of coefficients, and the number of constraints. We reduce only inequalities to preserve the affine dimension of the polyhedron. In practice, both simplifications are applied when the number of bits or constraints exceeds a given threshold that is significantly higher than the reduction level. The resulting hysteresis between exact computations and overapproximations gives cyclic dependencies time to stabilize.

11.3.1 Limiting the Number of Bits

We consider the ith constraint $a_i^T x \bowtie_i b_i$ of a polyhedron of the form $Ax + b \bowtie 0$, where a_i is a vector of the coefficients $a_{ij} \in \mathbb{Z}$ of A, $i = 1, \ldots, m$, $j = 1, \ldots, n$, $\bowtie$ is a vector of signs $\bowtie_i \in \{\leq, <, =\}$, and b is a vector of inhomogeneous coefficients $b_i \in \mathbb{Z}$. We assume that the a_{ij} and b_i have no common factor and that there are no redundant constraints. The goal is to find a new constraint $\alpha_i^T x \bowtie_i \beta_i$ with coefficients α_{ij} having less than z bits, i.e.,

$$|\alpha_{ij}|, |\beta_i| \leq 2^z - 1, \tag{11.3}$$

with the least overapproximation possible. Expressing the new coefficients in terms of a scaling factor $f > 0$, rounding errors r_{ij}, $|r_{ij}| \leq 0.5$ and an error r_i for the inhomogeneous term we get

$$\begin{aligned} \alpha_{ij} &= f a_{ij} + r_{ij}, \\ \beta_i &= f b_i + r_i. \end{aligned}$$

procedure *LimitConstraintBits*
Input: Polyhedron as a set of constraints $P = \{a_k^T x \bowtie_k b_k | k = 1, \ldots, m\}$,
index i to constraint to be limited, desired number of bits z
Output: new constraint $\alpha_i^T x \bowtie_i b_i$
$success := false$**;**
$f := min\{(2^z - 3/2)/|a_{kj}|, (2^z - 2)/|b_i| \mid j = 1, \ldots, n\}$**;**
while $\neg success$ **do**
for $j = 1, \ldots, n$ **do** $\alpha_{ij} := round(f a_{ij})$ **od;**
$q := \min_{x \in P} \alpha_i^T x$ **;**
if $\alpha_i = 0$ **or** $q = -\infty$ **then abort fi;**
$\beta_i := ceil(q)$**;**
if $|\beta_i| \leq 2^z - 1$ **then** $success := true$
else $f := min\{f/2 - 3/(4|a_{kj}|), (2^z - 2)/|\beta_i| \mid j = 1, \ldots, n\}$ **fi;**
od.

Figure 11.10: Algorithm for limiting the number of bits of a constraint

There is no a-priori bound on r_i, since it depends on the new direction α_i and the other constraints that define the polyhedron. With the bounds on the r_{ij}, we get $|f a_{ij} + r_{ij}| \leq 2^z - 1$, and get upper bounds on f using $|r_{ij}| \leq 0.5$ and, in the best case, we expect β_i to be close to $f b_i$. Since β_i must be rounded strictly upwards to guarantee conservativeness, we get $|r_i| \leq 1$ as an estimate:

$$f \leq (2^z - 3/2)/|a_{ij}|, \tag{11.4}$$
$$f \leq (2^z - 2)/|b_i|. \tag{11.5}$$

To predict the effects of rounding precisely is difficult and would lead to a mixed integer linear program, [2] so we employ a heuristic algorithm, shown in Fig. 11.10. Let $round(x)$ be a function that returns the next integer between x and zero, and $ceil(x)$ be a function that rounds to the next larger integer. First, we estimate f based on (11.4),(11.5), then we compute a new β_i using linear programming. If β_i has more than z bit, we decrease f and start over. The procedure is repeated until all coefficients $\alpha_{ij} = 0$, in which case the problem is infeasible. Note that it is not guaranteed that the new polyhedron is bounded. Figure 11.11 illustrates the basic scheme. The normal vector a_i of the constraint, shown in (a), is approximated by α_i, as shown in (b). Linear programming yields the inhomogeneous term q that makes the constraint tangent to the polyhedron, as in (c). Rounding of q yields β_i, and the polyhedron outlined in (d).

[2] The problem is much simpler if the polyhedron is given as a set of vertices instead of constraints, since the vertices only have to "snap" to the next points with the required number of bits. However, we try to avoid enumerating vertices since this is generally a very expensive operation in higher dimensions.

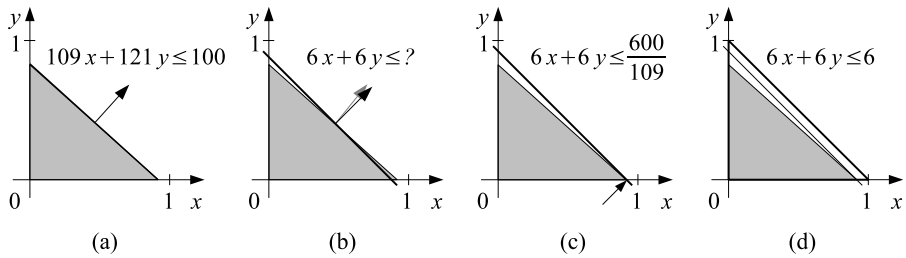


Figure 11.11: Scheme of limiting the number of bits

11.3.2 Limiting the Number of Constraints

A set of symbolic states is described by a linear predicate, the convex sub-predicates of which define convex polyhedra, which in turn are described by a set of constraints. The number of constraints usually increases with the number of iterations of a fixpoint computation, which forces us to limit this number to remain within a computationally feasible complexity. This reduction has been implemented in literature by means of, e.g., bounding boxes [BM99], or oriented rectangular hulls [SK03]. Instead, we propose to simply drop the least significant of the constraints, as this seems a good compromise in terms of accuracy and speed. In addition, the constraints in our applications are, as a whole, invariant with respect to the fixpoint computation. This invariance has a greater chance of being preserved as good as possible if we keep constraints instead of drawing up an entirely new set. As with limiting the number of bits, we usually chose to not limit equalities in order to preserve the affine dimension of the polyhedron. If an equality is to be limited, it must be replaced by two inequalities.

We measure the significance of a constraint based on a criterion *crit* that measures the the difference between the polyhedron with and without the constraint. Let P be a set of linear constraints describing a convex polyhedron, and $P^{\setminus i} = P \setminus \{a_i^T x \bowtie_i b_i\}$ be the polyhedron without it's ith constraint. Then the difference between the points contained P and $P^{\setminus i}$ is the polyhedron $P^{\neg i} = P^{\setminus i} \cup \{-a_i^T x \overline{\bowtie}_i - b_i\}$, where $(\bowtie_i, \overline{\bowtie}_i) \in \{(<,\leq),(\leq,<)\}$, obtained by simply replacing the ith constraint with its complement. It has less non-redundant constraints than P and is therefore preferable in the formulations below. We consider three methods:

1. volumetric: Let $V(P)$ be the volume of the points contained in P. Then $crit = V(P^{\setminus i}) - V(P) = V(P^{\neg i})$. Requires $P^{\neg i}$ to be bounded.

2. slack: Let $b_{max} = \max_x \; a_i^T x$ s.t. $x \in P^{\neg i}$. Then $crit = (b_{max} - b_i)/||a_i||$, i.e., the distance, measured in the direction of the constraint, between the points farthest apart in $P^{\neg i}$. Requires $P^{\neg i}$ to be bounded in the direction of a_i.

3. angle: $crit = -\max_{j \neq i} \; a_j^T a_i$. Measures the negative cosine of the closest angle between the normal vector of the ith constraint and all others.

We consider two general procedures of selecting the z most important out of m original constraints:

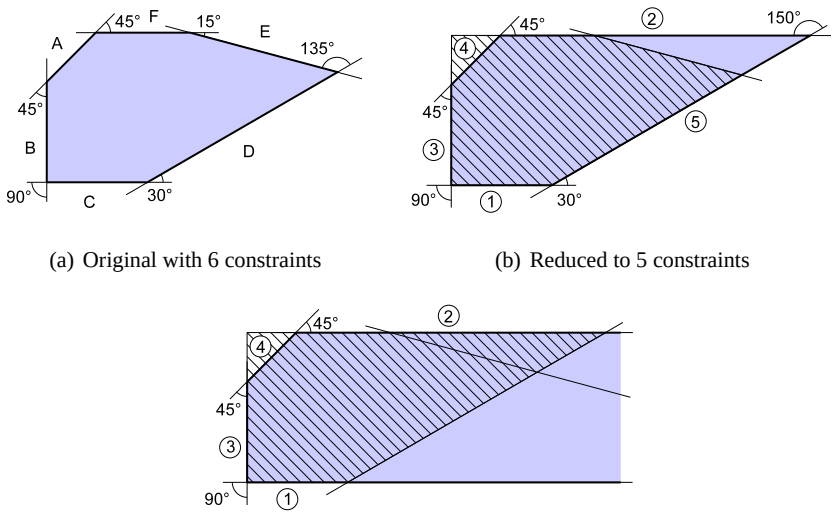
1. deconstruction: Starting from the entire set of constraints, drop the $m-z$ constraints with the least effect according to *crit*.

2. reconstruction: Starting from an empty set of constraints, add the z constraints with the greatest effect according to *crit*.

While deconstruction is more likely to preserve as much as possible of the original polyhedron, construction requires less iterations if $m > 2z$. The criteria based on volume and slack require the initial polyhedron to be bounded, for which one could use, e.g., the invariant of the location. The following example shall illustrate the difference between volumetric and angle criteria, and its potential unboundedness.

Example 11.1. *Consider the polyhedron shown in Fig. 11.12(a). It has 6 constraints A–F, whose angles with the neighbors are noted in the graph. In a volume based deconstruction with 5 constraints, constraint A is removed since that causes the smallest change in volume. The resulting polyhedron is shown hashed in Fig. 11.12(b). The angle based reconstruction with 5 constraints results in the shaded polyhedron in Fig. 11.12(b), where the constraints are labeled in the order they are chosen: First, an arbitrary initial constraint is chosen, say constraint C. The second choice is the constraint that has the largest angle with C, i.e., that is most opposed to it. In this case, this is constraint F, since it has an angle of* 180° *with C. The third choice is the one that is most opposed to both C and F, here constraint B because it has an angle of* 90° *with both C and F. The fourth constraint is A, with minimum angles of* 45°*, and the fifth is D with a minimum angle of* 30°*. Figure 11.12(c) shows the reduction to 4 constraints. Here the angle based method results in an unbounded polyhedron because constraint D is not chosen. An algorithm should take this possibility into account and test for boundedness.*

The construction method with an angle criterion was the fastest in our experiments. The angle calculations can be sped up by using a look-up table $\alpha(i,j)$ that maps an angle to every pair of constraints. This yields an algorithm of complexity $O(nm^2 + m^3)$, shown in Fig. 11.13, where C is the set of candidate constraints and H is the set of chosen constraints. It includes a test that preserves the boundedness of P. H is initialized with the set of equalities, which are not reduced to preserve the affine dimension of the polyhedron, and an arbitrary initial constraint. Here we choose the one with the smallest coefficients. In a while-loop, the constraint is chosen based on the best of the worst-cases, i.e., the smallest angle with the constraints in H. Since $a_j^T a_i$ is the cosine of the angle, choosing the smallest angle translates into maximizing $a_j^T a_i$. The constraint is added to H and removed from the candidates C, and the procedures is repeated until $|H| \geq z$ and the boundedness of P implies boundedness of H. This algorithm is in our implementation $\sim$1000$\times$ faster than a slack based deconstruction for limiting 400 constraints down to 32 in 4 dimensions.



(a) Original with 6 constraints

(b) Reduced to 5 constraints

(c) Reduced to 4 constraints

Figure 11.12: Example for limiting the number of constraints by volumetric deconstruction (hashed) and angle based reconstruction (shaded)

procedure *LimitConstraintsByAngle*

Input: Polyhedron P as a set of constraints $a_i^T x \bowtie_i b_i$, $i = 1, \ldots, m$,
desired number of constraints z

Output: Polyhedron H

for $i = 1, \ldots, m$**,** $j = 1, \ldots, m$**,** $j > i$ **do**

$\alpha(i, j) := a_i^T a_j$

od;

$H := \{a_k^T x \bowtie_k b_k \mid k = \text{argmin}_k(\max_j |a_{kj}|)\} \cup \{a_i^T x \bowtie_i b_i | \bowtie_i \in \{=\}\}$**;**

$C := P \setminus H$**;**

while $(|C| > 0 \wedge (|H| < z \vee (bounded(P) \wedge \neg bounded(H)))$ **do**

$j = \text{argmin}_j\ (max_i \alpha(i, j))$ **s.t.** $a_i^T x \bowtie_i b_i \in H, a_j^T x \bowtie_j b_j \in C$**;**

$H := H \cup \{a_j^T x \bowtie_j b_j\}$**;**

$C := C \setminus \{a_j^T x \bowtie_j b_j\}$

od.

Figure 11.13: Reconstructing a polyhedron with a limited number of constraints by angle prioritization

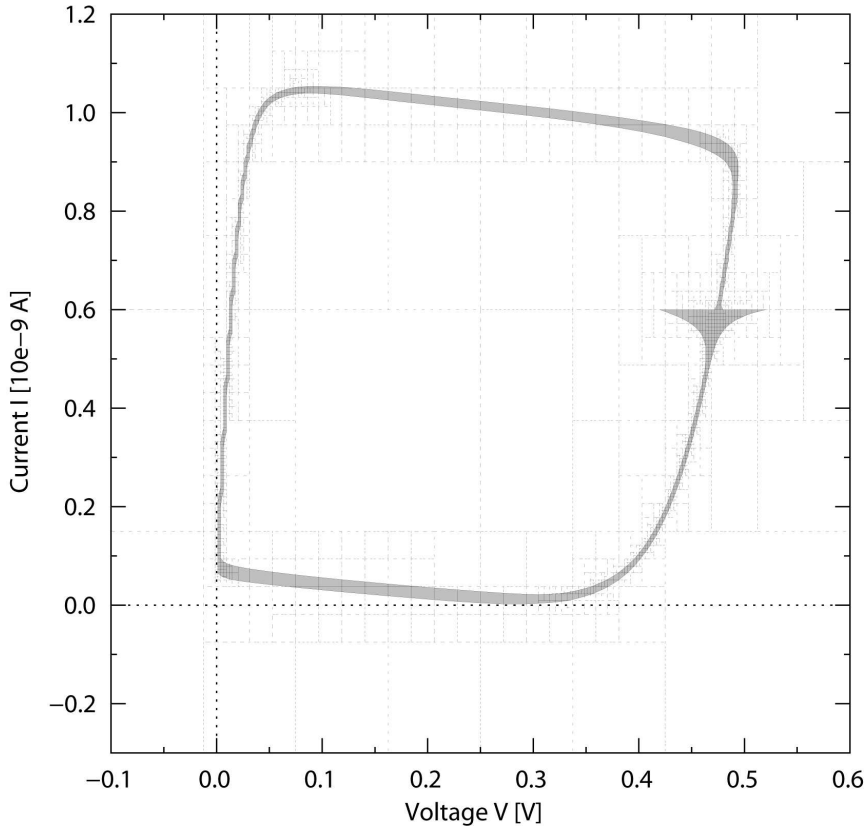


Figure 11.14: Reachable states of tunnel diode circuit in the V-I-plane, invariants dashed

11.3.3 Example: Tunnel-Diode Oscillator Circuit

Consider a tunnel-diode oscillator circuit as described in [GKR04]. It models the current I and the voltage drop V of a tunnel diode in parallel to the capacitor of a serial RLC circuit, which are in stable oscillation for the given parameters. The state equations are given by

$$\begin{aligned} \dot{V} &= 1/C(-I_d(V)+I), \\ \dot{I} &= 1/L(-V-1/G\cdot I+V_{in}), \end{aligned}$$

where $C = 1\ pF$, $L = 1\ \mu H$, $G = 5\ m\Omega^{-1}$, $V_{in} = 0.3\ V$, and the diode current

$$I_d(V) = \begin{cases} 6.0105V^3 - 0.9917V^2 + 0.0545V & \text{if} \quad V \leq 0.055, \\ 0.0692V^3 - 0.0421V^2 + 0.004V + 8.9579e{-}4 & \text{if} \quad 0.055 \leq V \leq 0.35, \\ 0.2634V^3 - 0.2765V^2 + 0.0968V - 0.0112 & \text{if} \quad 0.35 \leq V. \end{cases}$$

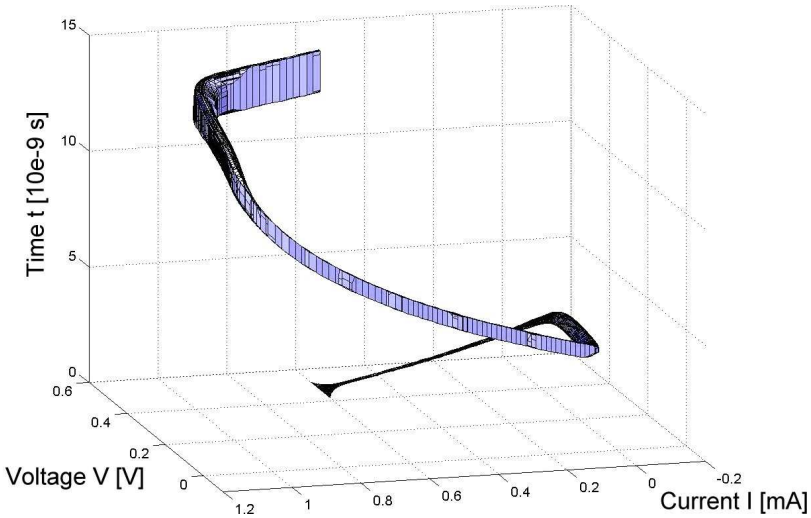
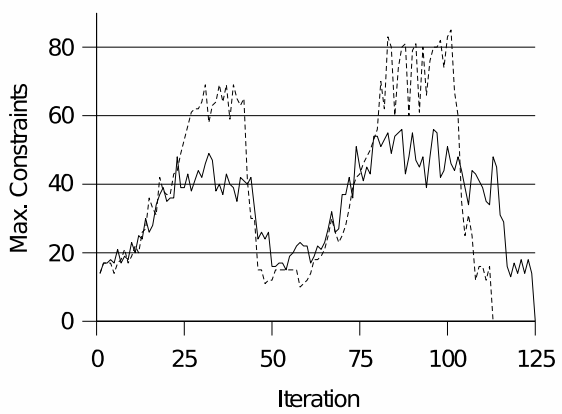


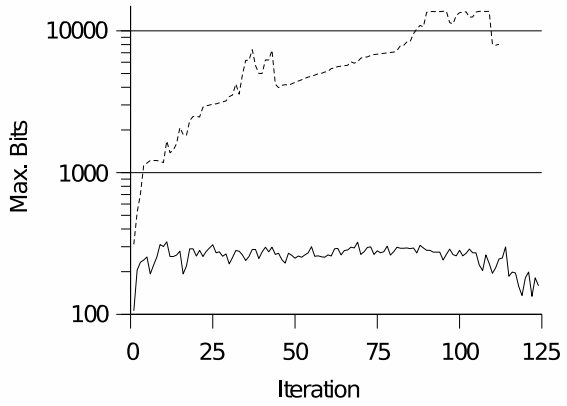
Figure 11.15: Reachable states of clocked tunnel diode circuit

The dynamics were approximated with LHA, similar to the approach in Sect. 11.2.1. Figure 11.14 shows the convex hull of the reachable states starting from states defined by $V \in [0.42V, 0.52V]$, $I = 0.6mA$. It also shows the invariants (dashed) generated by the partitioning algorithm using constraints $Cand = \{(V, 0.7/128, 0.7/16), (I, 1.5/128, 1.5/16)\}$, i.e., max. 128 partitions in both directions, and splitting criterion $(split_crit_3, split_crit_1)$ with $\sphericalangle_{min} = \arccos(0.99)$. The analysis with PHAVer took 52.63s and 55MB RAM, with the largest coefficient taking up 7352 bits and at most 7 constraints per polyhedron.

A stopwatch was added to the system to measure the cycle time, i.e., the maximum time it takes any state to cross the threshold $I = 0.6\mu A, V > 0.25V$ twice. For the clocked circuit, the number of bits and constraints grows rapidly and a more precise analysis, such as shown in Fig. 11.15 is only possible with limits on both. We compare the exact analysis for constraints $Cand = \{(V, 0.7/32, 0.7/16), (I, 1.5/32, 1.5/16)\}$ with an analysis limiting the bits to 16 when a threshold of 300 bits is reached, and a limit of 32 constraints at a threshold of 56. Figures 11.16(a) and 11.16(b) show a polynomial increase in the number of constraints, and an exponential increase of the number of bits in the new polyhedra found at each iteration. The analysis takes 979s (210MB) when exact, and 79s (39.6MB) when limited. At a more than tenfold increase in speed, the overapproximation is negligible and results in a cycle time estimate that is only 0.25 percent larger.



(a) Number of bits



(b) Number of constraints

Figure 11.16: Reduction in bits and constraints for the clocked tunnel diode circuit, exact (dashed) and with limits on bits and constraints (solid)

11.4 Related Work

Earlier attempts to improve over HyTech started with an algorithm specialized on rectangular automata, which are LHA whose linear predicates define multi-dimensional rectangles, was proposed in [PKWTH98] and implemented based on the HyTech engine. The set of reachable states is represented by the initial states and the intersection of the reachable states with the faces of the invariants. The set of reachable states in a location is then the convex hull of the faces and the initial set. The post operator is simplified by computing the minimum and maximum time it takes for a state to leave the invariant. A computation of these bounds can be done easily in a conservative way with limited precision, and allows the complexity to remain low. The successor sets on faces are overapproximated with hyperrectangles. While this approach has been successfully applied in the two-dimensional case [PSK99], the overapproximation with rectangular regions can become excessive for higher dimensions.

An improvement of this face-based approach was proposed in [Pre00], by generalizing from rectangular regions on the faces of invariants to arbitrary convex sets. The algorithm exploits the fact that the faces of invariants of rectangular automata are in the directions of the axes and orthogonal. The precision of the points defining the convex region is limited conservatively by replacing each vertice by the 2^n vertices (n being the number of dimensions) resulting from the possible combinations of componentwise rounding to the next lower and higher value. The algorithm operates solely with the vertices of polyhedra, whose number can increase exponentially with the dimension of the system, in addition to the increase introduced by limiting the precision.

A later improvement over HyTech by Henzinger et al. used interval arithmetic to speed up computations and reduce the complexity [HHMWT00]. However, the use of interval arithmetic can quickly lead to prohibitively large overapproximations.

While tools for timed automata usually use exact computations, there are no tools for hybrid systems apart from HyTech known to us that do so. The first HyTech prototype was based Mathematica and did not have any numerical restrictions, but it also was a factor 50–1000 times slower than the later version, which was written in C++ [HHWT95]. Our on-the-fly overapproximation essentially performs a partitioning of the automaton similar to the approach in [HHWT98], where Henzinger et al. approximate arbitrary nonlinear hybrid automata with LHA. We use a shared passed and waiting list in the reachability computation, a method proposed in [BDL$^+$01]. A detailed overview on algorithmic verification of hybrid automata can be found in [SSKE01].

The partitioning of hybrid automata according to the dynamics has been applied in literature to simplify the verification problem in various ways, although with different criteria. The formal foundation for such partitions was established in [HHWT98] by showing that a timed bisimulation exists under very general conditions. In [HHB02], hybrid automata are discretized completely, and discrete model checking is applied to the discrete abstractions. In [SK99], nonlinear hybrid automata are approximated with rectangular automata.

To our knowledge, the only other tool with algorithmic support for circular assume/guarantee reasoning is Mocha [HLQR99], see p. 62. It uses reactive modules, which can be used to approximate hybrid systems by discretization.

Chapter 12

Conclusions

The verification of hybrid systems suffers greatly from the state explosion problem, more so than discrete systems because of the drastic increase in complexity with the number of continuous variables. The best known solvers for boolean operations on linear predicates, which are interpreted as polyhedra in $\mathbb{R}^n$, are limited to around 20 variables for non-trivial problems, with computation times of several days even when using floating point arithmetic. This warrants the application of compositional methods, such as the well established framework for compositional reasoning based on simulation relations by Grumberg and Long, which we revisited and enhanced in Part I. As the main contribution of this thesis, we proposed its extension from discrete to hybrid automata, which required a new semantics to be able to handle shared variables. This extension takes two forms, presented in Parts II and III of this thesis:

In Part II we follow a definition of simulation for hybrid automata given by Henzinger [Hen96], using an equivalence relation to connect states and variables in the comparison. E.g., a variable $\hat{x}$ in a specification might represent a physical quantity with an identical representation in a variable x of the system, while another variable t in the specification might model timing properties that are not explicit in the system. An equivalence relation can be used to require that $\hat{x} = x$ in all states of the simulation relation, so that any property expressed in terms of $\hat{x}$ implies that the same property holds for x. This concept is very general in that arbitrary combinations of locations and variables can be related, and other properties, such as fairness, can be included in the equivalence relation. However, the structural information separating locations and variable valuations is lost during the composition of systems, so that arguing about shared variables is not possible on the semantic level. The application of this approach is therefore limited to systems without shared variables.

In Part III, we retain the separation of discrete locations and continuous variables in a novel semantics for hybrid systems based on *hybrid labeled transition systems*. We impose an Input/Output structure on the system and define equivalence based on the values of input- and output-variables. The semantics ensure that automata control the evolution of their output variables and encode the invariants of hybrid automata

in stutter transitions, thus making invariants expressible in simulation. Compared to Part II, we lose the ability to restrict the equivalence of locations, and only consider identity as equivalence of variables. Most importantly, we can show with these semantics that simulation based on timed transition system semantics is compositional for hybrid automata with unrestricted inputs, and for the fundamental class of linear hybrid automata, which can be used to approximate any hybrid automata arbitrarily close, and are accessible to algorithmic verification.

The results from applying simulation relations in compositional reasoning, and the above extensions to hybrid systems, are summarized in the following sections.

12.1 Discrete Systems

Simulation relations are an intuitive concept for comparing systems. They are readily and intuitively applicable in compositional reasoning, as proposed by Grumberg and Long [GL91]. Originally, simulation was defined only between automata with identical alphabets. We propose an extension of this simulation concept to arbitrary alphabets called Σ-simulation. It is consistent with the classic definition, since it coincides with simulation in the case of identical alphabets. Most importantly, it retains the precongruence properties of simulation and we showed that it is the largest such extension of simulation. The freedom to compare automata of arbitrary alphabets enables simpler proofs and slightly smaller models for specifications. Conveniently, Σ-simulation can be implemented effortlessly in classic simulation frameworks and tools, since simulation and Σ-simulation easily transpose with the help of *Chaos*-automata, which simply introduce self-loops in every location. This translation has inspired more compact simulation proofs and tautologies and should be of general relevance to the verification of discrete systems.

Our compositional framework follows in its basic structure that of Grumberg and Long. It differs through the use of *Chaos*-automata and Σ-simulation, and extends it with some stronger theorems, such as the decomposition of the specification. For circular assume/guarantee-reasoning, we propose a novel proof rule. Contrary to a rule by Henzinger et al. proposed in [HQRT02] we do not require receptiveness. We believe it to be the most permissive rule possible in a simulation framework, although this is yet to be proven.

12.2 Hybrid Systems with Discrete Interaction

Thomas Henzinger proposed in [Hen96] to define simulation for hybrid systems based on their timed transition system (TTS) semantics, and to include an equivalence relation for that identifies which states in the system should correspond to which in the specification. We were able to show that, if the systems share no variables, the TTS-semantics and the parallel composition operator commute, and all properties of

labeled transition systems can be transferred directly to hybrid automata. As a consequence, TTS-simulation for such hybrid systems is compositional, and the compositional framework of Part I can be applied as long as the simulation relations are contained in the equivalence relations associated with the comparison of automata. We extend the major rules for compositional reasoning from Part I and obtain the restrictions imposed by the equivalence relations.

12.3 Hybrid Systems with Continuous Interaction

In the TTS-semantics of Part II, locations and variables are amalgamated to a state of a labeled transition system, and it is impossible to extract variable valuations from states of a TTS. This prompted us to introduce an extension of labeled transition systems, called hybrid labeled transition systems (HLTSs), which retains the structure of locations and variables. We use HLTSs to define semantics for hybrid input/output-automata (HIOAs), and require in simulation that input- and output variables of the same name in automata under comparison must have the same values. Internal variables of the systems in the comparison are considered to be unrelated. We also propose a corresponding path semantics, and show consistency with TTS-semantics.

Simulation is defined based on the TTS-semantics, and we show that this definition is consistent with and weaker than simulation based on traces. Because the TTS-semantics abstract from the continuous activities of variables by existential quantification, TTS-simulation is in general not compositional in the presence of shared variables. We show that compositionality holds for two important classes of hybrid systems: hybrid automata with unrestricted inputs, and linear hybrid automata with convex invariants. Since any hybrid automaton can be approximated arbitrarily close with a linear hybrid automaton, this opens a way to verify many hybrid systems compositionally by approximating the components and the specification with linear hybrid automata. Our definition of unrestricted inputs implies that the inputs can not directly affect the activities of the controlled variables, and thus excludes dynamics such as $\dot{x} = u$. They can, however, be used to model sampled feedback systems, since for those the value of the input is relevant only at discrete time instants.

For other classes, the TTS-semantics must be applied in compositional reasoning before the composition operator, which introduces an abstraction from the continuous interaction of the components. With respect to the continuous dynamics, this corresponds to assuming open inputs, and can also lead to a violation of nonconvex invariants. E.g., a differential equation $\dot{x} = u$ with open input u means that $x(t)$ can take arbitrary values for $t > 0$. This is a gross overapproximation that prevents the compositional analysis of, e.g., systems with continuous feedback. This overapproximation can be remedied to an extend by assuming bounds on the inputs or their derivatives (especially in A/G-reasoning).

Assume/Guarantee-reasoning seems particularly warranted and useful when dealing with hybrid systems. For one, even if the abstraction contains just one continuous

variable less than the concrete system, it can cut the computation costs by several orders of magnitude. Secondly, since the continuous state space has a metric, properties like invariants often find a compact representation. This is particularly interesting when one is able to give a *proper* specification, i.e., one that is independent of the actual implementation.

12.4 PHAVer

The computational complexity of hybrid systems remains a challenging problem, and is hard to predict. In the general case, neither reachability nor simulation are guaranteed to terminate in finite time. This prompted us to obtain experimental results as early as possible during the work on this topic. Started in October of 2002, the algorithms have evolved into a tool called PHAVer (Polyhedral Hybrid Automaton Verifyer). It was, to our knowledge, pioneering in two respects, and was initially more a proof of concept than a full-fledged implementation of a ready-to-use tool. Firstly, it was the first application of exact arithmetic with unbounded representation in hybrid systems verification, and it was totally open how the computation engine – the Parma Polyhedral Library (PPL), which had been released only a few months earlier – would compare to other tools or be useful at all. Secondly, it was the first implementation for checking simulation for hybrid systems, as well as assume-guarantee reasoning, and we chose to first implement a simple but rigorous approach to obtain sound results, rather than optimizing algorithms and data-structures from the start.

The results with respect to the performance of exact arithmetic, are rather positive, in no small amount thanks to the quality of the PPL. The application of exact arithmetic and not-necessarily-closed polyhedra has paid off not only by side-stepping numerical problems, but with a separation of concerns: Phenomena of convergence, overapproximation and termination can be clearly identified and counteracted correspondingly, rather than being intertwined beyond recognition through numerical limitations. On several occasions, we would have wrongly identified problems to be of numerical origin, and consequently might have given up on solving them rather than tracing their roots. Naturally, the complexity of linear predicates usually increases drastically during the course of a computation. Similarly to a limited-precision approach, we have to resort to overapproximation in those cases, but have the advantage that we can do so intelligently and with guaranteed conservativeness. For this we proposed algorithms for limiting the number of bits and coefficients of a convex linear predicate.

The results with our crude implementation of simulation checking are mixed. On the bright side, the simulation check is actually faster than reachability for a few examples, and sometimes it terminates quickly while exact reachability (without limiting bits and constraints) does not terminate at all. On the down side, our implementation suffers greatly from the complexity of computations with higher dimensional polyhedra, and is orders of magnitudes slower than a reachability analysis of the same property. To check simulation for $P \preceq Q$, the current, simple, version computes a sim-

ulation relation R in the state space of $S_P \times S_Q$, i.e., over the product of the state spaces of P and Q. A transition in P is checked by intersecting the target states with the simulation relation R, which yields an operation in $S_P \times S_P \times S_Q$. This is dramatically more expensive than a computation in the state space of P alone. A further limitation is the costly difference operation in the course of the fixed-point computation of a simulation relation. It requires very expensive simplification procedures for the polyhedral sets to keep the number of convex polyhedra at an acceptable level. This simplification is implemented very crudely, and could probably be much improved. In Sect. 6.3.2 we mentioned a number of improvements to alleviate these problems, but their implementation was beyond the timeframe of this thesis.

12.5 Future Research

Simulation Recall that we could show with Σ-simulation that the parallel composition of two specifications corresponds to their logical conjunction. It should be possible to also define disjunction and complement operators, and thus obtain a boolean algebra. Such a construction could be applied, e.g., in equation solving, controller synthesis, and to generate assumptions for assume/guarantee reasoning algorithmically.

We did not take into account weak simulation, i.e., silent transitions that are undetectable through simulation relations. This, however, is be essential to compare approximations of hybrid automata with linear hybrid automata that have not the exact same state partitioning. According to our formalism, this weak simulation should still be strong with respect to environment labels, and only convex invariants should be partitioned with silent transitions. This does not prevent the partitioning of the automaton with labels with respect to which the simulation is strong, yet prevents the specification from having partitions that are not in the system.

Compositional Reasoning The successful application of compositional reasoning to a more general class of hybrid automata requires a more detailed model of the interaction than the TTS-semantics used in this thesis. Further work should try to establish a simulation concept that also describes the trajectories, e.g., by providing bounds for the derivative. This can be generalized to a paradigm of augmented states, in which the trajectories are described by bounded parameters over a family of functions.

PHAVer On the implementation side several possible improvements were outlined in Sect. 6.3.2, and they can be expected to yield several orders of magnitude improvement over the current simulation algorithms.

To be able to check simulation between LHA-overapproximations of nonlinear hybrid automata, silent transitions, i.e., checking of weak simulation, should be implemented. While this is straightforward if done in a naive way, it is a challenge to devise an efficient implementation that avoids the redundant computation of time elapse sets over several locations connected with silent transitions.

In applying our on-the-fly overapproximation to hybrid systems with affine dynamics it has become evident that an effective partitioning is the key factor to success. Currently, all reachable states are partitioned, independently of whether they are relevant to the specification or not. Significant advances in both reachability and simulation checking should be possible by iterative refinement, in the case of reachability with respect to a set of forbidden states. Initially, we assume a coarse partitioning. A first analysis identifies which states are violating the specification or are forbidden in this coarse partition. Through backtracking or backwards reachability analysis, one can identify a subset of the reachable states that contains the violating runs, and refine the partition only for this subset. The analysis is repeated in the forward direction, this time restricted to the runs that are violating, and with a smaller partition in only those states. In this manner, the number of partitions under examination can decrease as the partitions get smaller, while in a conventional analysis, this is never the case. This approach is closely related to Counter Example Guided Abstraction Refinement (CEGAR) in [CFH$^+$03].

Appendix A

PHAVer Language Overview

We have tried to construct a textual input language that is as user friendly as possible, while keeping the parser simple. In the syntax, we have borrowed extensively from the creators of HyTech [HHWT97], since their language is intuitively understandable. The following sections describe the syntax of PHAVer's representation of automata, states, relations etc., and brief descriptions of the analysis commands, followed by a section on the user-definable parameters.

A.1 General

Comments are preceded by either `//`, `--`, or enclosed in `/*` ... `*/`. An *identifier* is a letter plus any combination of letters, digits and the characters _ (underscore) and ~ (tilde), where ~ is designated for joining the identifiers of locations of composed automata. A number can be given in floating point format, e.g., `3.14` or `6.626e-34`, or as a fraction, e.g., `9/5`. Note that numbers are internally represented as exact rationals, and no conversion to binary floating point format takes place (which would lead to rounding errors). E.g., the input `0.1` is parsed and represented in PHAVer as $1/10$, while a 64-bit floating point representation of 0.1 would actually be the number

$$0.1000000000000000055511151231257827021181583404541015625.$$

Commands, constants, parameters and automata can occur in an arbitrary sequence. A command is terminated by `;` (semi-colon).

A.2 Constants

Constants are defined in the form *identifier* `:=` *expression*`;` where *expression* is any combination of expressions, identifiers and numbers with `+`,`-`,`/`,`*`,`(`,`)`.

A.3 Data Structures

There are four types of data structures that can be assigned to identifiers: linear predicates, sets of symbolic states, symbolic relations and automata. A *linear expression* is specified over an arbitrary set of variables, numbers and constants that can be combined using `+`,`-`,`/`,`*`,`(`,`)` as long as it yields a linear expression as defined in Def. 5.5. I.e., it is not allowed to multiply two variables, or divide by a variable, and the attempt to do so will result in an error message. A linear constraint is a combination of two linear expressions with one of the signs `<`,`>`,`<=`,`>=`,`==`. A *convex linear predicate* is given as a conjunction of linear constraints that are joined by `&` (ampersand). A *linear predicate* is a disjunction of convex linear predicates joined by `|`. Brackets `(`,`)` can be used to avoid ambiguities. A linear predicate can be assigned to a variable in the form *identifier* = *linear predicate*`;`.

A *symbolic state* is a combination of a location name and a linear predicate, joined by `&`, e.g., `start & x>0 & y==0`. A *set of symbolic states* is a list of symbolic states, joined by `,` (commata). A set of symbolic states of an automaton *aut* is assigned to a variable in the form *identifier* = *aut*`.{` *set of symbolic states* `};` Symbolic relations are returned by the simulation relation algorithms. There are currently no provisions for specifying relations. Identifiers can be assigned to other identifiers simply using `=`.

In the following, let *var_ident* be an identifier defining a variable, *loc_ident* a name for a location, *label_ident* an identifier defining a synchronization label. An automaton with identifier *aut* is specified in the following form:

```
automaton aut
   state_var: var_ident, var_ident, ...;
   input_var: var_ident, var_ident, ...;
   parameter: var_ident, var_ident, ...;
   synclabs:  lab_ident, lab_ident, ...;
   loc loc_ident: while invariant wait { derivative };
       when guard sync label_ident do {trans_rel} goto loc_ident;
       when ...
   loc loc_ident: while ...
end
```

The *invariant* and the *guard* are linear predicates over the state and input variables and the parameters. The *derivative* definition depends on the dynamics:

- For "linear" (LHA) dynamics, it is a convex linear predicate over the state variables. E.g., `0 <= x & x < 1` for $\dot{x} \in [0,1)$.
- For affine dynamics, it is a convex linear predicate over the variables and their derivatives. The non-differentiated variables are indicated by `'` (single quote), e.g., `x == -2 * x'` for $\dot{x} = -2x$.

Note that parameter uncertainties can be incorporated by using inequalities. A lin-

ear predicate *trans_rel* specifies the continuous transition relation μ, where the post-transition value of the variable is indicated by `'` (single quote). State variables that are not changed by the transition must be specified, e.g., `x'==x & y'==y`. The reset of a variable to 0 would be defined, e.g., by $z' == 0$. Automata are composed using `&` (ampersand), e.g., `comp_aut = aut1 & aut2;`

A.4 Commands

PHAVer provides commands for computing reachable sets of states and simulation relations, plus a number of commands for the manipulation and output of data structures. In the following list, square brackets [] are used to indicate optional arguments. *identifier* is used to denote the identifier for an arbitrary object, *predicate_ident* for a linear predicate, *state_ident* for a set of symbolic states, *rel_ident* for a symbolic relation and *aut_ident* for an automaton. Let *state_or_rel_ident* stand for either a set of symbolic states or a relation. Let *state_list* be an explicit comma separated list of symbolic states, e.g., `start & t==0, stop & t==1`. Recall that objects can be copied with an assignment *new_identifier* = *old_identifier*`;`

A.4.1 General

- `echo "`*text*`";`
  Displays *text* and starts a new line.

- `who;`
  Displays a list of identifiers currently in the memory.

- *identifier*`.print(`[*file_name*][`,`*method*]`);`
  If *file_name* is specified, writes a representation of *identifier* to the file *file_name*, otherwise to the standard output. An optional integer *method* determines the format:

  - `0`: (default) Location names and linear predicates are produced in textual form.
  - `1`: Output the linear predicates as a sequence of linear constraints in floating point form. Equalities $\phi = 0$, where ϕ is some linear predicate, are converted to $\phi \geq 0 \wedge -\phi \geq 0$. The coefficients of a constraint $\sum a_i x_i + b \bowtie 0$ are output separated by spaces, one constraint per line. Convex predicates are separated by a blank line. No location information is given. This form can be used for output with polyhedral visualization packages. The order of variables is the same as in the list provided by the automaton output.
  - `2`: Output the linear predicates as a sequence of vertices in floating point form. The vertices belonging to a convex predicate are separated by a blank

line. No location information is given. This form can be used for output with plotting tools like `graph`. If 2-dimensional, the points are in counter-clockwise order and represent a closed line for each convex predicate, i.e., the last point is equal to the first. The order of variables is the same as in the list provided by the automaton output.

A.4.2 Reachability Analysis

- *state_ident*=*aut_ident*`.reachable;`
  Returns the set of states reachable in the automaton *aut_ident* from the initial states.

- *state_ident1*=*aut_ident*`.reachable(`*state_ident2*`);`
  Returns the set of states reachable in the automaton *aut_ident* from the states in *state_ident2*.

- *state_ident1*=*aut_ident*`.reachable_stop(`*state_ident2*`);`
  Computes the set of reachable states, but stops as soon as a state in *state_ident2* is found. Returns the states in *state_ident2* that were found to be reachable before termination, i.e., the ones of the last iteration.

A.4.3 Simulation Checking

- *rel_ident*=`get_sim(`*aut_ident1*,*aut_ident2*`);`
  Returns a simulation relation for *aut_ident1*$\preceq$*aut_ident2*.

- `is_sim(`*aut_ident1*,*aut_ident2*`);`
  Computes a simulation relation and displays whether *aut_ident1*$\preceq$*aut_ident2*.

- `is_bisim(`*aut_ident1*,*aut_ident2*`);`
  Computes a simulation relation and displays whether *aut_ident1*$\preceq$*aut_ident2*.

- `ag_sim(`*aut_ident1*,*aut_ident2*,*aut_ident3*,*aut_ident4*`);`
  Computes a simulation relation using assume/guarantee reasoning and displays whether *aut_ident1* $||$ *aut_ident2* $\preceq$ *aut_ident3* $||$ *aut_ident4*.

A.4.4 Refinement

- *aut_ident*`.set_refine_constraints(((`*lin_expr1*,δ_{1min},δ_{1max}`),`
  `(`*lin_expr2*,δ_{2min},δ_{2max}`),\dots);`
  Defines the refinement constraints used in subsequent analyses. A location will be split by a constraint of the form *lin_expr1*$\leq c$, where c is the center of the location with respect to the linear expression. δ_{1min} and δ_{1max} define the minimum and maximum extent of every location in the refinement process. The constraints are prioritized according to the refinement parameters.

A.4.5 Manipulation Commands

- *identifier*`.remove(`*var_ident,var_ident,...*`);`
  Existential quantification over the specified variables.

- *identifier*`.project(`*var_ident,var_ident,...*`);`
  Existential quantification over all except the specified variables.

- *identifier*`.get_parameters(`*bool*`);`
  Performs existential quantification over state and input variables, i.e., non-parameters. A boolean parameter *bool* specifies the quantification over locations:

  – `false`: Disjunction, the parameters are common to all locations. E.g., compute the set of reachable states, intersect it with a set of desired states, and get the parameters for which all desired states are reachable with option `false`.

  – `true`: Conjunction, the parameters occur in any of the locations. E.g., compute the set of reachable states, intersect it with a set of forbidden states, and get the parameters for which any of the forbidden states are reachable with option `true`.

- *predicate_ident=state_or_rel_ident*`.loc_union;`
  Returns the states unified over the locations.

- *predicate_ident=state_or_rel_ident*`.loc_intersection;`
  Returns the states intersected over the locations.

- *identifier1*`.contains(`*identifier2*`);`
  Writes whether the object *identifier2* is contained in the object *identifier1* of the same type to the standard output.

- *identifier*`.is_empty;`
  Writes whether the object *identifier* is empty to the standard output.

- *rel_ident1=rel_ident2*`.inverse;`
  If *rel_ident2* is a relation R, then *rel_ident1* is assigned R^{-1}.

- *rel_ident1=rel_ident2*`.project_to_first;`
  If *rel_ident2* is a relation $R(p,q)$, then *rel_ident1* is assigned the relation defined by $R' = \{p | \exists q : R(p,q)\}$.

- *aut_ident*`.add_label(`*lab_ident*`);`
  Adds the label *lab_ident* to the alphabet of the automaton *aut_ident*. Can be used to add dedicated labels for refinement to an existing model.

A.5 Parameters

The following is a summary of the parameters used in PHAVer. A parameter is defined in the form *identifier* = *value*;. The default setting is given in brackets, and the type is boolean unless specified otherwise.

A.5.1 General

- `ELAPSE_TIME` (true): Can be used to switch off the time-elapse operator. Useful for the analysis of purely discrete systems, but the speed-up is modest.

A.5.2 Reachability Analysis

- `REACH_MAX_ITER` (0): Integer specifying the maximum number of iterations used, i.e., the number of discrete transitions explored. Only active if > 0.

- `CHEAP_CONTAIN_RETURN_OTHERS` (true): Determines the type of containment test used:

  – false: exact, i.e., a convex polyhedron p is considered contained in a non-convex polyhedron q if the difference $p \setminus q$ is empty.

  – true: a convex polyhedron p is considered contained in a non-convex polyhedron $q = q_1 \cup \ldots \cup q_n$ (a union of convex polyhedra) if there is a convex polyhedron q_i in the union that contains it, i.e., $\exists i \in \{1, \ldots, n\} : q \subseteq q_i$. This method is generally faster than exact testing, although it results in more polyhedra.

- `REACH_STOP_AT_FORB` (false): When checking for reachability of a set of forbidden states, stop as soon as a forbidden state is encountered.

- `USE_CONVEX_HULL` (false): Use convex-hull overapproximations. Highly recommended when using on-the-fly refinement, and usually a good idea.

- `REACH_STOP_USE_CONVEX_HULL_ITER` (1000000000): Integer specifying the maximum number of iterations for which the convex-hull overapproximation is used. Can be set to a lower value to improve termination.

- `REACH_USE_BBOX` (false): Causes the overapproximation of the post-transition states with a bounding box. Can be used to force termination, but usually leads to excessive over-approximation.

- `REACH_USE_BBOX_ITER` (1000000000): Integer specifying the frequency n (a number of iterations) with which the bounding-box overapproximation is used. It is only applied at one iteration, then followed by n iterations with the normal

setting. Can be set to a lower value to improve termination. Note that it is independent of `REACH_USE_BBOX`.

- `REACH_ONLY_EXPLORE` (false): Toggles a special *exploration* mode: There is no testing if newly reached states are contained in previous ones. Terminates only if the number of iterations is set with `REACH_MAX_ITER`.

- `CONSTRAINT_BITSIZE` (0): Integer specifying the number of bits used in constraints, i.e., in the polyhedral computations. Equalities are not affected. If a constraint can not be specified with that amount of bits, a error is thrown. Only active if > 0.

- `REACH_BITSIZE_TRIGGER` (0): Integer threshold for limiting the number of bits used. The bits are reduced as specified in `CONSTRAINT_BITSIZE` only if they exceed this threshold. Can significantly improve termination and reduce the overapproximation, usually set to $10--30\times$ the limit.

- `REACH_STOP_USE_BITSIZE` (1000000000): Integer specifying the maximum number of iterations for which the number of bits are constrained. Can be set to a lower value to improve termination.

- `LIMIT_CONSTRAINTS_METHOD` (1): Integer specifying the method with which the number of constraints is reduced:

  - 0: those constraints are preserved, whose extent is maximal if they were omitted (slow).
  - 1: those constraints are preserved that maximize the angle with all other constraints (very fast).

- `REACH_CONSTRAINT_LIMIT` (0): Integer specifying the maximum number of constraints allowed in a convex polyhedron. Exceeding polyhedra will be overapproximated as specified by `LIMIT_CONSTRAINTS_METHOD`. When using convex-hull overapproximations, the limiting is performed before the time-elapse operator, so that the resulting number of constraints can be higher. Usually set to at least 2^n if n is the number of variables. Only active if > 0.

- `REACH_CONSTRAINT_TRIGGER` (0): Integer threshold for limiting the number of constraints. A convex polyhedron is reduced to `REACH_CONSTRAINT_LIMIT` constraint once it exceeds this threshold. Can significantly improve termination, usually set to 2–3$\times$ the constraint limit. Boundedness of the polyhedron is preserved, with more constraints if necessary. Only active if > 0.

- `TP_CONSTRAINT_LIMIT` (0): Integer specifying the maximum number of constraints used for describing the derivative. Exceeding derivative predicates will

be overapproximated as specified by `LIMIT_CONSTRAINTS_METHOD`. Boundedness of the polyhedron is preserved, with more constraints if necessary. Only active if > 0.

A.5.3 Simulation Checking

- `PRIME_R_WITH_REACH` (true): Initialize the simulation relation with the reachable states of $P||Q$.
- `USE_CONVEX_HULL_FOR_PRIMING` (true): Use convex-hull reachability for the initialization if `PRIME_R_WITH_REACH` is true.
- `PRIME_R_WITH_DISCRETE_REACH` (true): Overapproximating initialization of the simulation relation with the locations of $P||Q$ that are reachable by discrete transitions.
- `STOP_AT_BAD_STATES` (true): Stop as soon as bad states are encountered. Only useful if `PRIME_R_WITH_REACH=true`.
- `SHOW_BAD_STATES` (false): Output bad states as they are encountered.
- `SIM_SIMPLIFY_R` (true): Simplify the simulation relation, i.e., remove redundant polyhedra and join convex unions where possible, after each difference operation. Costly, but usually indispensable.

A.5.4 Refinement

- `TIME_POST_ITER` (0): Integer specifying how many iterations are performed between reachable states and restricting the derivative to those reachable states. Higher numbers improve the accuracy of refined dynamics, but numbers > 0 require that the derivative is re-computed every time the cell is examined, which significantly slows down the analysis.
- `REFINE_CHECK_TIME_RELEVANCE` (true): Eliminate refinement transitions that are never crossed by timed transitions and are therefore irrelevant.
- `REFINE_DERIVATIVE_METHOD` (0): Integer determining how the set of derivatives is determined:
  - 0: convex hull of the derivatives occurring in the location,
  - 1: bounding box of method 0 (faster),
  - 2: center of method 0. This method is useful for creating sample runs of the system for a single state. The reachable set for a single state can usually be determined very fast, and can be compared with results of, e.g., simulation tools.

- `REFINE_PRIORITIZE_REACH_SPLIT` (false): Prioritize constraints that have reachable states strictly on both sides.
- `REFINE_PRIORITIZE_ANGLE` (false): Prioritize constraints according to the maximum angle spanned by the derivative in the resulting locations.
- `REFINE_SMALLEST_FIRST` (false): Prioritize constraints according to their extent in the location. True corresponds to smallest first.
- `REFINE_DERIV_MAXANGLE` (1): Floating point number a_{max}. A location is only refined if $a_{max} > cos(\alpha_{max})$, where α_{max} is the max. angle between any two derivative vectors in the location. A value smaller than 1 results in a refinement based loosely on the "curvature" of the derivative, typically between $0.85 - 0.99$.

Bibliography

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995. A preliminary version appeared in the Proceedings of the 11th International Conference on Analysis and Optimization of Systems: Discrete-Event Systems (ICAOS), Lecture Notes in Control and Information Sciences 199, Springer-Verlag, 1994, pp. 331-351.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [GNRR93], pages 209–229.

[AD92] R. Alur and D.L. Dill. The theory of timed automata. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Real-Time: Theory in Practice*, Mook, The Netherlands, June 1991, volume 600 of *LNCS*, pages 45–73. Springer, 1992.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994. (preliminary versions appeared in *Proc. 17th ICALP*, LNCS 443, 1990, and *Real Time: Theory in Practice*, LNCS 600, 1991).

[ADE+01] Rajeev Alur, Thao Dang, Joel M. Esposito, Rafael B. Fierro, Yerang Hur, Franjo Ivančić, Vijay Kumar, Insup Lee, Pradyumna Mishra, George J. Pappas, and Oleg Sokolsky. Hierarchical hybrid modeling of embedded systems. In Thomas A. Henzinger and Christoph M. Kirsch, editors, *EMSOFT 2001: First International Workshop on Embedded Software, Tahoe City, CA, USA, October 8–10, 2001*, volume 2211 of *LNCS*, pages 14–31. Springer, 2001.

[ADI02] Rajeev Alur, Thao Dang, and Franjo Ivančić. Reachability analysis of hybrid systems via predicate abstraction. In Tomlin and Greenstreet [TG02], pages 35–48.

[AG00] Rajeev Alur and Radu Grosu. Modular refinement of hierarchic reactive machines. In *POPL 2000, Proceedings of the 27th ACM SIGPLAN-SIGACT on Principles of Programming Languages, January 19-21, 2000, Boston, Massachusetts, USA*, pages 390–402. ACM Press, 2000.

[AGLS01] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional refinement of hierarchical hybrid systems. In Di Benedetto and Sangiovanni-Vincentelli [DSV01], pages 33–48.

[AH97] Rajeev Alur and Thomas A. Henzinger. Modularity for timed and hybrid systems. In Mazurkiewicz and Winkowski [MW97], pages 74–88.

[AHH96] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996. A preliminary version appeared in the Proceedings of the 14th Annual Real-Time Systems Symposium (RTSS), IEEE Computer Society Press, 1993, pp. 2-11.

[AHS96] Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors. *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *LNCS*. Springer, 1996.

[AK96] Rajeev Alur and Robert P. Kurshan. Timing analysis in COSPAN. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *LNCS*, pages 220–231. Springer, 1996.

[AL93] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(15):73–132, 1993.

[AL95] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 17(3):507–534, May 1995.

[All03] Roger Allan. Electronics will inspire safer, fun-to-drive cars. *Electronic Design*, June 2003. http://www.elecdesign.com/Articles/Index.cfm?ArticleID=5044.

[BCC98] Sergey Berezin, Sérgio Campos, and Edmund M. Clarke. Compositional reasoning in model checking. In Roever et al. [RLP98], pages 81–102.

[BDL+01] Gerd Behrmann, Alexandre David, Kim G. Larsen, Oliver Möller, Paul Pettersson, and Wang Yi. UPPAAL - present and future. In *Proc. of 40*th *IEEE Conference on Decision and Control*. IEEE Computer Society Press, 2001.

[BLN03] Dirk Beyer, Claus Lewerentz, and Andreas Noack. Rabbit: A tool for BDD-based verification of real-time systems. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *LNCS*, pages 122–125. Springer, 2003.

[BM99] Alberto Bemporad and Manfred Morari. Verification of hybrid systems via mathematical programming. In Vaandrager and van Schuppen [VvS99], pages 31–45.

[BP95] Bard Bloom and Robert Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24(3):189–220, June 1995.

[BRRS96] Amar Bouali, Annie Ressouche, Valérie Roy, and Robert de Simone. The FC2Tools set. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, volume 1102 of *LNCS*, pages 441–445. Springer, 1996.

[BRZ04] Roberto Bagnara, Elisa Ricci, and Enea Zaffanella. PPL: The Parma Polyhedra Library, April 2004. Available at http://www.cs.unipr.it/ppl/.

[BRZH02] Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In Manuel V. Hermenegildo and German Puebla, editors, *Static Analysis: Proc. of the 9th Int. Symposium*, volume 2477 of *LNCS*, pages 213–229, Madrid, Spain, 2002. Springer.

[BS98] Sébastien Bornot and Joseph Sifakis. On the composition of hybrid systems. In Thomas A. Henzinger and Shankar Sastry, editors, *Hybrid Systems: Computation and Control, First International Workshop, HSCC'98, Berkeley, California, USA, April 13-15, 1998, Proceedings*, volume 1386 of *LNCS*, pages 49–63. Springer, 1998.

[Cat04] Adam Cataldo. Control algorithms for soft walls. Master's Report Technical Memorandum UCB/ERL M03/42, University of California, Berkeley, CA 94720, January 2004.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Proceedings of the Workshop on Logic of Programs*, Yorktown Heights, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

[CEG+01] Darren D. Cofer, Eric Engstrom, Robert P. Goldman, David J. Musliner, and Steve Vestal. Applications of model checking at Honeywell Laboratories. In Matthew B. Dwyer, editor, *Model Checking Software, 8th International SPIN Workshop, Toronto, Canada, May 19-20, 2001, Proceedings*, volume 2057 of *LNCS*, pages 296–303. Springer, 2001.

[CFH+03] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Olaf Stursberg, and Michael Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In Hubert Garavel and John Hatcliff, editors, *Proc. 9th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, volume 2619 of *LNCS*, pages 192–207. Springer, 2003.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. The MIT Press, Cambridge, MA, USA, 1999.

[CJ00] Pierre Collette and Cliff B. Jones. Enhancing the tractability of rely/guarantee specifications in the development of interfering operations. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction*, chapter 10, pages 277–307. MIT Press, 2000.

[CK03] Alongkrit Chutinan and Bruce H. Krogh. Computational techniques for hybrid system verification. *IEEE Trans. on Automatic Control*, 48:64–75, 2003.

[Cou93] Costas Courcoubetis, editor. *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *LNCS*. Springer, 1993.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, pages 148–165. Springer-Verlag, 2001.

[DSV01] Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *LNCS*. Springer, 2001.

[FI04] Ansgar Fehnker and Franjo Ivancic. Benchmarks for hybrid systems verification. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings*, volume 2993 of *LNCS*, pages 326–341. Springer, 2004.

[FKM93] Jean-Claude Fernandez, Alain Kerbrat, and Laurent Mounier. Symbolic equivalence checking. In Courcoubetis [Cou93], pages 85–96.

[Fre23] Gottlob Frege. Logische Untersuchungen. Dritter Teil: Gedankengefüge. *Beiträge zur Philosophie des Deutschen Idealismus*, 3(1):36–51, 1923. English translation in [Fre77].

[Fre77] Gottlob Frege. Compound thoughts. In P. Geach and N. Black, editors, *Logical Investigations*. Blackwells, Cambridge, UK, 1977.

[Fre05] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control (HSCC'05), Mar. 9–11, 2005, Zürich, CH*, volume 2289 of *LNCS*. Springer, 2005. PHAVer is available at http://www.cs.ru.nl/~goranf/.

[Fuk04] Komei Fukuda. Polyhedral computation FAQ, June 2004. ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports/polyfaq041121.pdf.

[Fur03] Carlo A. Furia. Compositional proofs for real-time modular systems. Laurea degree thesis, Politecnico di Milano, 2003.

[FV98] Kathi Fisler and Moshe Y. Vardi. Bisimulation minimization in an automata-theoretic verification framework. In Ganesh Gopalakrishnan and Phillip J. Windley, editors, *Formal Methods in Computer-Aided Design, Second International Conference, FMCAD '98, Palo Alto, California, USA, November 4-6, 1998, Proceedings*, volume 1522 of *LNCS*, pages 115–132. Springer, 1998.

[GKR04] Smriti Gupta, Bruce H. Krogh, and Rob A. Rutenbar. Towards formal verification of analog designs. In *Proc. IEEE Intl. Conf. on Computer-Aided Design (ICCAD-2004), Nov. 7–11, 2004, San Jose CA (USA)*, 2004.

[GL91] Orna Grumberg and David E. Long. Model checking and modular verification. In Jos C. M. Baeten and Jan Friso Groote, editors, *CONCUR '91, 2nd International Conference on Concurrency Theory, Amsterdam, The Netherlands, August 26-29, 1991, Proceedings*, volume 527 of *LNCS*. Springer, 1991.

[Gla90] Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *LNCS*, pages 278–297. Springer, 1990.

[GNRR93] Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors. *Hybrid Systems*, volume 736 of *LNCS*. Springer, 1993.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27-30 July 1996*, pages 278–292. IEEE Computer Society Press, 1996. An extended version appeared in Verification of Digital and Hybrid Systems (M.K. Inan, R.P. Kurshan, eds.), NATO ASI Series F: Computer and Systems Sciences, Vol. 170, Springer-Verlag, 2000, pp. 265-292.

[HHB02] Walter Hartong, Lars Hedrich, and Erich Barke. On discrete modeling and model checking for nonlinear analog systems. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *LNCS*, pages 401–413. Springer, 2002.

[HHK95] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 453–462. IEEE Computer Society Press, 1995.

[HHMWT00] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-Toi. Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23-25, 2000, Proceedings*, volume 1790 of *LNCS*, pages 130–144. Springer, 2000.

[HHWT95] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. HYTECH: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, page 56. IEEE Computer Society, 1995.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, December 1997.

[HHWT98] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):540–554, 1998.

[HKR97] Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. In Mazurkiewicz and Winkowski [MW97], pages 273–287.

[HLFE02] Ralf Huuck, Ben Lukoschus, Goran Frehse, and Sebastian Engell. Compositional verification of continuous-discrete systems. In Sebastian Engell, Goran Frehse, and Eckehard Schnieder, editors, *Modelling, Analysis and Design of Hybrid Systems*, volume 279 of *Lecture Notes in Control and Information Sciences*, pages 225–244. Springer, 2002.

[HLQR99] Thomas A. Henzinger, Xiaojun Liu, Shaz Qadeer, and Sriram K. Rajamani. Formal specification and verification of a dataflow processor array. In *International Conference on Computer-Aided Design, ICCAD'99, November 7-11, 1999, San Jose, CA, USA*, pages 494–499. ACM, 1999.

[HMP01] Thomas A. Henzinger, Marius Minea, and Vinayak Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In Di Benedetto and Sangiovanni-Vincentelli [DSV01], pages 275–290.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.

[Ho95] Pei-Hsin Ho. *Automatic Analysis of Hybrid Systems*. PhD thesis, Cornell University, August 1995. Technical Report CSD-TR95-1536.

[Hoa85] Charles A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, 1985.

[Hoo93] Jozef Hooman. A compositional approach to the design of hybrid systems. In Grossman et al. [GNRR93], pages 121–148.

[Hoo98] Jozef Hooman. Compositional verification of real-time applications. In Roever et al. [RLP98], pages 276–300.

[HPR94] Nicolas Halbwachs, Yann-Eric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In Baudouin Le Charlier, editor, *Static Analysis, First International Static Analysis Symposium, SAS'94, Namur, Belgium, September 28-30, 1994, Proceedings*, volume 864 of *LNCS*, pages 223–237. Springer, 1994.

[HPWT01] Thomas A. Henzinger, Joerg Preussig, and Howard Wong-Toi. Some lessons from the hytech experience. In *Proceedings of the 40th Annual Conference on Decision and Control (CDC'01)*, pages pp. 2887–2892. IEEE Press, 2001.

[HQR98] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. You assume, we guarantee: Methodology and case studies. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *LNCS*, pages 440–451. Springer, 1998.

[HQR00] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. Decomposing refinement proofs using assume-guarantee reasoning. In *Proceedings of the 2000 International Conference on Computer-Aided Design, ICCAD 2000, November 5-9, 2000, San Jose, California, USA*, pages 245–252. IEEE, 2000.

[HQRT02] Thomas A. Henzinger, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. An assume-guarantee rule for checking simulation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(1):51–64, 2002.

[HS96] Hans Hüttel and Sandeep Shukla. On the complexity of deciding behavioural equivalences and preorders. Report RS-96-39, BRICS, Department of Computer Science, University of Aarhus, Aarhus, Denmark, October 1996.

[HWT96a] Thomas A. Henzinger and Howard Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In Alur et al. [AHS96], pages 377–388.

[HWT96b] Thomas A. Henzinger and Howard Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, number 1165 in LNCS, pages 265–282. Springer Verlag, 1996.

[Iva03] Franjo Ivancic. *Modeling and Analysis of Hybrid Systems*. PhD thesis, University of Pennsylvania, Philadelphia, PA, December 2003.

[Jen99] Henrik Ejersbo Jensen. *Abstraction-Based Verification of Distributed Systems*. PhD thesis, Aalborg University, June 1999.

[JLS00] Henrik Ejersbo Jensen, Kim Guldstrand Larsen, and Arne Skou. Scaling up Uppaal – automatic verification of real-time systems using compositionality and abstraction. In Mathai Joseph, editor, *FTRTFT 2000:*

6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, September 20–22, 2000, Pune, India, volume 1926 of *LNCS*, pages 19–30. Springer, 2000.

[JM01] Ranjit Jhala and Kenneth L. McMillan. Microarchitecture verification by compositional model checking. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *LNCS*, pages 396–410. Springer, 2001.

[Jon81] Cliff B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University Computing Laboratory, June 1981. Printed as: Programming Research Group, Technical Monograph 25.

[Jon83] Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.

[Jon94] Bengt Jonsson. Compositional specification and verification of distributed systems. *ACM Trans. Program. Lang. Syst.*, 16(2):259–303, 1994.

[Jon03] Cliff B. Jones. Wanted: a compositional approach to concurrency. In Annabelle McIver and Carroll Morgan, editors, *Programming Methodology*, pages 1–15. Springer, 2003.

[Kel76] Robert M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, July 1976.

[KEPS99] Stefan Kowalewski, Sebastian Engell, Jörg Preußig, and Olaf Stursberg. Verification of logic controllers for continuous plants using timed condition/event-system models. *Automatica*, 35(3):505–518, mar 1999. Special Issue on Hybrid Systems.

[Kow96] Stefan Kowalewski. *Modulare diskrete Anlagenmodellierung zum systematischen Steuerungsentwurf*. Dissertation, Lehrstuhl für Anlagensteuerungstechnik, Universität Dortmund, Aachen, Germany, 1996.

[KPSY93] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs a class of decidable hybrid systems. In *Proceedings of "Workshop on Theory of Hybrid Systems", Lyngby, Denmark, June 1992*, volume 736 of *LNCS*, pages 179–208. Springer, 1993.

[KSF+99] Stefan Kowalewski, Olaf Stursberg, Martin Fritz, Holger Graf, Ingo Hoffmann, Joerg Preussig, Manuel Remelhe, Silke Simon, and Heinz

Treseler. A case study in tool-aided analysis of discretely controlled continuous systems: The two tanks problem. In Panos J. Antsaklis, Wolf Kohn, Michael D. Lemmon, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems V*, volume 1567 of *LNCS*, pages 163–185. Springer, 1999.

[LLL00] Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of TCAS. *Proceedings of the IEEE, Special Issue on Hybrid Systems*, 2000.

[Lon93] David E. Long. *Model Checking, Abstraction and Compositional Verification*. PhD thesis, Carnegie Mellon University, July 1993.

[LPY01] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32:231–253, 2001.

[LS95] Nancy A. Lynch and Roberto Segala. A comparison of simulation techniques and algebraic techniques for verifying concurrent systems. *Journal of Formal Aspects of Computing Science*, 7(3):231–265, 1995. An extended abstract appears in Proceedings of the Second North American Process Algebra Workshop, Cornell University, NY, 1993.

[LSV01] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata revisited. In Di Benedetto and Sangiovanni-Vincentelli [DSV01], pages 403–417.

[LSV03] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

[LSVW96] Nancy A. Lynch, Roberto Segala, Frits W. Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In Alur et al. [AHS96].

[LSW97] Kim Guldstrand Larsen, Bernhard Steffen, and Carsten Weise. Continuous modeling of real-time and hybrid systems: From concepts to tools. *International Journal on Software Tools for Technology Transfer*, 1(1-2):64–85, 1997.

[LT87] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, August 10-12, 1987, Vancouver, British Columbia, Canada*, pages 137–151. ACM, 1987.

[LV95] Nancy A. Lynch and Frits W. Vaandrager. Forward and backward simulations, I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., San Fransisco, California, 1996.

[MC81] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, July 1981.

[McM99] Kenneth L. McMillan. Circular compositional reasoning about liveness. In Laurence Pierre and Thomas Kropf, editors, *Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings*, volume 1703 of *LNCS*, pages 342–345. Springer, 1999.

[McM00] Kenneth L. McMillan. A methodology for hardware verification using compositional model checking. *Science of Computer Programming*, 37(1–3):279–309, 2000.

[Mil71] Robin Milner. An algebraic definition of simulation between programs. In David C. Cooper, editor, *Proc. of the 2nd Int. Joint Conference on Artificial Intelligence. London, UK, September 1971*, pages 481–489. William Kaufmann, British Computer Society, 1971.

[MK99] Jeff Magee and Jeff Kramer. *Concurrency: state models & Java programs*. John Wiley & Sons, Inc., 1999.

[MK01] Jeff Magee and Jeff Kramer. LTSA, Labelled Transition System Analyser v2.2, January 2001. http://www.doc.ic.ac.uk/~jnm/.

[MMP92] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Proc. REX Workshop "Real-Time: Theory in Practice"*, pages 447–484, London, UK, 1992. Springer-Verlag.

[MRTT53] T.S. Motzkin, H. Raiffa, G.L. Thompson, and R.M. Thrall. The double description method. In *Contributions to the Theory of Games Vol. 2*. Princeton University Press, 1953.

[MW97] Antoni W. Mazurkiewicz and Józef Winkowski, editors. *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *LNCS*. Springer, 1997.

[NOSY92] Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. An approach to the description and analysis of hybrid systems. In Grossman et al. [GNRR93], pages 149–178.

[NSY93] Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. From atp to timed graphs and hybrid systems. *Acta Inf.*, 30(2):181–202, 1993. An earlier

version appeared in *Proc. REX Workshop Real-Time: Theory in Practice*, The Netherlands, June 1991, LNCS 6000, Springer-Verlag.

[NT00] Kedar S. Namjoshi and Richard J. Trefler. On the completeness of compositional reasoning. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *LNCS*, pages 139–153. Springer, 2000.

[OSY94] Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. Using abstractions for the verification of linear hybrid systems. In David L. Dill, editor, *Computer Aided Verification, 6th International Conference, CAV '94, Stanford, California, USA, June 21-23, 1994, Proceedings*, volume 818 of *LNCS*, pages 81–94. Springer, 1994.

[Par81] D. M. R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5th *GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

[PDH99] Corina S. Pasareanu, Matthew B. Dwyer, and Michael Huth. Assume-guarantee model checking of software: A comparative case study. In Dennis Dams, Rob Gerth, Stefan Leue, and Mieke Massink, editors, *Theoretical and Practical Aspects of SPIN Model Checking, 5th and 6th International SPIN Workshops, Trento, Italy, July 5, 1999, Toulouse, France, September 21 and 24 1999, Proceedings*, volume 1680 of *LNCS*, pages 168–183. Springer, 1999.

[PKWTH98] Jörg Preußig, Stephan Kowalewski, Howard Wong-Toi, and Thomas A. Henzinger. An algorithm for the approximative analysis of rectangular automata. In *Proceedings of the Fifth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, number 1486 in LNCS, pages 228–240. Springer-Verlag, 1998.

[PLS98] G.eorge J. Pappas, Gerardo Lafferriere, and Shankar Sastry. Hierarchically consistent control systems. In *Proc. of the 37th IEEE Conference in Decision and Control*, Tampa, FL, December 1998.

[Pre00] Jörg Preußig. *Formale Überprüfung der Korrektheit von Steuerungen mittels rektangulärer Automaten*. PhD thesis, Schriftenreihe des Lehrstuhls für Anlagensteuerungstechnik Band 4/2000, Universität Dortmund, Shaker Verlag, 2000. (in German).

[PSK99] Jörg Preußig, Olaf Stursberg, and Stefan Kowalewski. Reachability analysis of a class of switched continuous systems by integrating rectangular approximation and rectangular analysis. In Vaandrager and van Schuppen [VvS99], pages 209–222.

[PT87] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[PT98] Hong Peng and Sofiéne Tahar. A survey on compositional verification. Technical report, Dept. of Electrical and Computer Engineering, Concordia University, Montreal, Canada, November 1998.

[Raj99] Sriram K. Rajamani. *New Directions in Refinement Checking*. PhD thesis, University of California, Berkeley, September 1999.

[RBH+01] Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Number 54 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, November 2001.

[RLP98] Willem P. de Roever, Hans Langmaack, and Amir Pnueli, editors. *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997. Revised Lectures*, volume 1536 of *LNCS*. Springer, 1998.

[Roe98] Willem-Paul de Roever. The need for compositional proof systems: A survey. In Roever et al. [RLP98], pages 1–22.

[Rus01] John Rushby. Formal verification of McMillan's compositional assume-guarantee rule. Technical report, SRI International Computer Science Laboratory, September 2001.

[SAGG+93] Jørgen F. Søgaard-Andersen, Stephen J. Garland, John V. Guttag, Nancy A. Lynch, and Anna Pogosyants. Computer-assisted simulation proofs. In Courcoubetis [Cou93], pages 305–319.

[SBB+04] Jörg Schöneberg, Uwe Berndt, George Blau, Jens Friedemann, Hans-W. Hempelmann, Eberhard Krupper, Heinrich-H. Niebaum, Hans Peters, Johann Reuß, Dieter Ritschel, Karsten Severin, Frank Stahlkopf, and Axel Thiel. Status report AX001-1/-2/02. Technical report, Bundesstelle für Flugunfalluntersuchung (German Federal Bureau of Aircraft Accidents Investigation), Braunschweig, Germany, May 2004. http://www.bfu-web.de/berichte/02_ax001dfr.pdf.

[SIRS96] Sandeep K. Shukla, Harry B. Hunt III, Daniel J. Rosenkrantz, and Richard Edwin Stearns. On the complexity of relational problems for finite state processes (extended abstract). In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Automata, Languages*

and Programming, 23rd International Colloquium, ICALP'96, Paderborn, Germany, 8-12 July 1996, Proceedings, volume 1099 of *LNCS*, pages 466–477. Springer, 1996.

[SK91] Ramavarapu S. Sreenivas and Bruce H. Krogh. On condition/event systems with discrete state realizations. *Discrete Event Dynamic Systems: Theory and Applications*, 1(2):209–236, 1991.

[SK99] Olaf Stursberg and Stefan Kowalewski. Approximating switched continuous systems by rectangular automata. In *Proc. 5th European Control Conference, Karlsruhe*, 1999.

[SK03] Olaf Stursberg and Bruce H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*, volume 2623 of *LNCS*, pages 482–497. Springer, 2003.

[SL02] Y. Shang and M. D. Lemmon. The controlled composition of hybrid automata through the use of inner viability kernels. In *Proc. 15th Int. Symp. Mathematical Theory of Networks and Systems*, Notre Dame, IN, USA, August 2002.

[SS01] A. J. van der Schaft and J. M. Schumacher. Compositionality issues in discrete, continuous, and hybrid systems. *Int. Journal of Robust and Nonlinear Control*, 11(5):417–434, April 2001.

[SSKE01] B. I. Silva, O. Stursberg, B. H. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Proc. 40th Conference on Decision and Control (CDC'01)*, December 2001.

[TAKB96] Serdar Taşiran, Rajeev Alur, Robert P. Kurshan, and Robert K. Brayton. Verifying abstractions of timed systems. In *International Conference on Concurrency Theory*, pages 546–562, 1996.

[Tar55] Alfred Tarski. A lattice theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 1955.

[TC01] Li Tan and Rance Cleaveland. Simulation revisited. In Tiziana Margaria and Wang Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2031 of *LNCS*, pages 480–495. Springer, 2001.

[TG02] Claire Tomlin and Mark R. Greenstreet, editors. *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25-27, 2002, Proceedings*, volume 2289 of *LNCS*. Springer, 2002.

[Tom96] Claire Tomlin. Verification of an air traffic management protocol using hytech. Course Project for EE290A, taught by Prof. T. A. Henzinger, Spring 1996, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1996.

[TP01] Paulo Tabuada and George J. Pappas. Hybrid abstractions that preserve timed languages. In Di Benedetto and Sangiovanni-Vincentelli [DSV01], pages 501–514.

[TP02] Herbert G. Tanner and George J. Pappas. Simulation relations for discrete-time linear systems. In *Proc. of the 15th IFAC World Congress on Automatic Control*, pages 1302–1307, Barcelona, Spain, 2002.

[TPL01] Paulo Tabuada, George J. Pappas, and Pedro U. Lima. Compositional abstractions of hybrid control systems. In *Proc. of the 40th IEEE Conference on Decision and Control, Orlando, Florida.*, 2001.

[TPL02] Paulo Tabuada, George J. Pappas, and Pedro U. Lima. Composing abstractions of hybrid systems. In Tomlin and Greenstreet [TG02], pages 436–450.

[TPL04] Paulo Tabuada, George J. Pappas, and Pedro Lima. Compositional abstractions of hybrid control systems. *Discrete Event Dynamic Systems*, 14(2):203–238, 2004.

[vG01] Rob J. van Glabbeek. What is branching time semantics and why to use it? In Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Current Trends in Theoretical Computer Science, Entering the 21th Century*, pages 469–479. World Scientific, 2001.

[VH96] Jan Vitt and Jozef Hooman. Assertional specification and verification using PVS of the steam boiler control system. In Jean-Raymond Abrial, Egon Börger, and Hans Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *LNCS*, pages 453–472. Springer, 1996.

[VK88] P. Varaiya and A. B. Khurzhanski, editors. *Discrete Event Systems: Models and Applications, Proc. International Institute for Applied Systems Analysis (IIASA) Workshop, Sopron, Hungary*, volume 103 of *LNCS*, New York, NY, 1988. Springer.

[VV01] Mahesh Viswanathan and Ramesh Viswanathan. Foundations for circular compositional reasoning. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *LNCS*, pages 835–847. Springer, 2001.

[VvS99] Frits W. Vaandrager and Jan H. van Schuppen, editors. *Hybrid Systems: Computation and Control, Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29-31, 1999, Proceedings*, volume 1569 of *LNCS*. Springer, 1999.

[Wil97] Doran Wilde. Polylib - a library of polyhedral functions, 1997. www.ee.byu.edu/~wilde/polyhedra.html.

[WL97] Carsten Weise and Dirk Lenzkes. Weak refinement for modal hybrid systems. In Oded Maler, editor, *Hybrid and Real-Time Systems, International Workshop. HART'97, Grenoble, France, March 26-28, 1997, Proceedings*, volume 1201 of *LNCS*, pages 316–330. Springer, 1997.

[XS98] Qiwen Xu and Mohalik Swarup. Compositional reasoning using the assumption-commitment paradigm. In Roever et al. [RLP98], pages 565–583.

[Zwi89] Job Zwiers. *Compositionality, Concurrency and Partial Correctness – Proof Theories for Networks of Processes and Their Relationship*, volume 321 of *LNCS*. Springer, 1989.

Samenvatting

Ten gevolge van de hand over hand toenemende integratie van microprocessoren in bijna alle elektrische apparaten, toepassingen en technische processen, speelt software een steeds belangrijkere rol bij de besturing van fysieke systemen. De interactie tussen software en fysieke apparaten en processen kan leiden tot complex, gecombineerd continu-discreet gedrag dat de analytische mogelijkheden tart van zowel de klassieke regeltheorie, die zich primair richt op systemen met zuiver continue tijd en waarden, als van de technieken uit de informatica, die betrekking hebben op discrete tijd en waarden. Systemen met gecombineerd continu-discreet gedrag worden *hybride systemen* genoemd. Veel veiligheidskritische toepassingen, zoals systemen voor luchtverkeersleiding en "drive-by-wire", vertonen hybride gedrag. Het doel van *formele verificatie*, dat wil zeggen het wiskundig modelleren van een systeem en het formeel bewijzen van zijn eigenschappen, is op ontwerpniveau te garanderen dat een systeem voldoet aan zijn specificatie (en dus veilig is). Terwijl formele verificatie een standaardgereedschap is geworden bij het ontwerp van discrete systemen zoals digitale circuits, wordt de toepassing voor hybride systemen gehinderd door de inherente complexiteit van discreet-continu gedrag. Verificatie is slechts toepasbaar bij relatief eenvoudige systemen omdat de rekenkosten exponentieel toenemen met het aantal componenten en toestandsvariabelen. Dit fenomeen wordt aangeduid als het *toestandsexplosie-probleem*. Alhoewel toestandsexplosies zich ook voordoen bij zuiver discrete systemen, hebben hybride systemen bovendien te kampen met de complexiteit van hoger dimensionale continue dynamica, die dramatisch toeneemt met het aantal toestandsvariabelen.

Om de verificatie van grotere systemen mogelijk te maken, breiden we in dit proefschrift abstractie en compositioneel redeneren, twee fundamentele benaderingen die succesvol worden toegepast in het discrete domein, uit naar de wereld van hybride systemen. In een *abstractie* van een system wordt informatie weggegooid die irrelevant is voor het bewijzen van gewenste eigenschappen. Het resulterende vereenvoudigde model dient de geldigheid van deze eigenschappen te behouden. Een *compositionele analyse* van een systeem onderzoekt de delen van een systeem op zodanige wijze dat eigenschappen van het gehele systeem kunnen worden afgeleid uit eigenschappen van de delen. Een bijzonder effectieve manier van compositioneel bewijzen is aanname/garantie-redeneren (assume/guarantee, A/G). Hierbij wordt de specificatie van een deelsysteem gebruikt als een aanname die het gedrag van andere deelsystemen beperkt, hetgeen leidt tot circulaire of kettingregelachtige bewijzen.

Om formeel aan te tonen dat een model inderdaad een abstractie is van een ander model, construeren we een *simulatierelatie* tussen de toestanden van de modellen. Een toestand *simuleert* een andere indien hetzelfde (of meer) gedrag mogelijk is. Intuïtief is dit behoudend in de zin dat als iets niet kan plaatsvinden in de abstractie, het ook niet kan plaatsvinden in het concrete systeem. Dezelfde aanpak kan gebruikt worden om een systeem en zijn specificatie te vergelijken; hierbij correspondeert het systeem met het verfijnde model en de specificatie met het abstracte model. Of een compositionele analyse altijd geldig is hangt af van de manier waarop de notie van simulatie is gedefinieerd. Niet alle soorten simulaties zijn berekenbaar voor hybride systemen. Het voornaamste doel dat met dit proefschrift wordt beoogd is compositionaliteit aan te tonen voor een notie van simulatie die berekenbaar is voor een relevante klasse van hybride systemen. Het proefschrift is onderverdeeld in drie delen. In het eerste deel wordt een uitbreiding beschreven van de theorie van simulaties voor discrete systemen van Grumberg and Long [GL91] met een verbeterde notie van simulatie en additionele bewijsregels voor compositioneel redeneren. In het tweede deel wordt aangetoond dat dit raamwerk direct toepasbaar is op hybride systemen zonder gedeelde variabelen door gebruik te maken van gelabelde transitiesystemen vergelijkbaar met die van Henzinger [Hen96]. In het derde deel worden hybride systemen met gedeelde variabelen in een invoer/uitvoer setting behandeld. We introduceren een gewijzigde semantiek die ons in staat stelt om compositionaliteit aan te tonen voor twee fundamentele klassen van hybride systemen: die met onbeperkte invoer en die met stuksgewijze constante begrenzingen van de afgeleiden en lineaire beperkingen. De laatste klasse is in het bijzonder relevant aangezien ieder hybride systeem met elke gewenste nauwkeurigheid benaderd kan worden door systemen uit deze (beslisbare) klasse.

Hieronder geven we een meer gedetailleerde samenvatting van de resultaten in de drie delen van het proefschrift.

Discrete Systemen Simulatierelaties vormen een intuïtief concept voor het vergelijken van systemen. Zoals aangetoond door Grumberg and Long [GL91] zijn ze gemakkelijk en intuïtief toepasbaar ten behoeve van compositioneel redeneren.

Oorspronkelijk waren simulaties alleen gedefinieerd tussen automaten met dezelfde alfabetten. Wij stellen een uitbreiding voor van deze notie van simulatie, Σ-simulatie genaamd, naar willekeurige alfabetten. Deze uitbreiding is consistent met de klassieke definitie, aangezien ze overeenstemt met simulatie in het geval de alfabetten gelijk zijn. De belangrijkste verdienste is dat de precongruentie-eigenschappen van simulaties gehandhaafd worden; we laten zelfs zien dat het de grootste uitbreiding van simulaties is die hier aan voldoet. De vrijheid om willekeurige automaten te vergelijken leidt tot eenvoudiger bewijzen en kleinere specificaties. Erg handig is dat Σ-simulatie probleemloos geïmplementeerd kan worden in klassieke raamwerken en gereedschappen voor simulaties: simulatie en Σ-simulatie kunnen eenvoudig getransponeerd worden met behulp van *Chaos*-automaten, die simpelweg zelf-lussen introduceren in iedere locatie. Deze vertaling leidt tot meer compacte bewijzen van simulaties en tautologie-

ën, en is derhalve van algemeen belang voor de verificatie van discrete systemen.

Ons compositionele raamwerk volgt in zijn basisstructuur dat van Grumberg en Long. Het verschilt door het gebruik van *Chaos*-automaten en Σ-simulaties, en breidt het uit met enkele meer krachtige stellingen, onder andere over de decompositie van specificaties. Voor circulair aanname/garantie-redeneren stellen we een nieuwe bewijsregel voor. Een aanname/garantie-regel is een compositionele bewijsregel die meer stringente specificaties toelaat door de modules van het systeem te combineren met aannames over het gedrag van hun omgeving, en levert daarmee in het algemeen een kleinere verzameling problemen. Indien deze aannames op een circulaire manier afhangen van de specificatie, is de regel alleen geldig indien aan additionele voorwaarden is voldaan die deze circulariteit doorbreken. Wij presenteren aanname/garantie-condities voor simulatierelaties en laten zien dat deze condities noodzakelijk en voldoende zijn voor het bestaan van een simulatierelatie. In tegenstelling tot een regel die door Henzinger et al. is beschreven in [HQRT02] eisen wij geen ontvankelijkheid ("receptiveness").

Hybride Systemen met Discrete Interactie Thomas Henzinger stelde in [Hen96] voor om simulaties voor hybride systemen te definiëren op basis van hun getimede transitiesystemen (Timed Transition System, TTS) semantiek, en met een equivalentierelatie aan te geven welke toestanden van het systeem dienen te corresponderen met welke toestanden in de specificatie. Wij laten zien dat, indien de systemen geen variabelen delen, de TTS-semantiek en de parallelle compositie-operator commuteren en dat in dit geval alle eigenschappen van gelabelde transitiesystemen direct overgedragen kunnen worden naar hybride automaten. Als gevolg hiervan is TTS-simulatie compositioneel voor zulke hybride systemen, en het compositionele raamwerk voor discrete systemen kan toegepast worden, mits de simulatierelaties bevat zijn in de equivalentierelaties die geassocieerd zijn met de te vergelijken automaten. We breiden de belangrijkste regels voor compositioneel redeneren uit van discrete naar hybride systemen en leiden beperkingen af die opgelegd dienen te worden aan de equivalentierelaties. Het concept van simulatie met equivalentie is zeer algemeen in de zin dat willekeurige combinaties van locaties en variabelen kunnen worden gerelateerd en andere eigenschappen, zoals fairness, opgenomen kunnen worden in de equivalentierelatie. Indien echter normale gelabelde transitiesystemen gebruikt worden als semantische fundering, raakt de structurele informatie die locaties en valuaties van variabelen scheidt verloren bij de compositie van systemen, waardoor redeneren over gedeelde variabelen niet mogelijk is op semantisch niveau. De toepassing van deze aanpak is derhalve beperkt tot systemen zonder gedeelde variabelen.

Hybride Systemen met Continue Interactie In de TTS-semantiek van Henzinger worden locaties en variabelen samengesmolten tot toestanden van een gelabeld transitiesysteem, en is het onmogelijk om valuaties van variabelen te extraheren uit de toestanden van een TTS. Dit bracht ons ertoe een uitbreiding te introduceren van gela-

belde transitiesystemen, hybride gelabelde transitiesystemen (HLTS) genaamd, die de structuur van locaties en variabelen behoudt. We gebruiken HLTSs om de semantiek van hybride invoer/uitvoer-automaten (Hybrid Input/Output Automata, HIOA) te definiëren en eisen in de definitie van simulatie dat de invoer- en uitvoervariabelen van de automaten onder beschouwing gelijk zijn. We nemen aan dat elke geregelde variabele die geen deel uitmaakt van de uitvoer niet in relatie staat met de variabele van het andere systeem. We definiëren ook een corresponderende paden-semantiek en tonen de consistentie hiervan aan met onze TTS-semantiek.

We stellen een notie van simulatie voor die gebaseerd is op de TTS-semantiek, en tonen aan dat deze consistent is met en zwakker dan simulatie op basis van traces. Aangezien de TTS-semantiek abstraheert van de continue activiteit van variabelen door existentiële kwantificatie, is TTS-simulatie in het algemeen niet compositioneel in aanwezigheid van gedeelde variabelen. Het is ons echter gelukt om compositionaliteit aan te tonen voor twee belangrijke deelklassen van hybride systemen: hybride automaten met onbeperkte invoer, en lineair hybride automaten met convexe invarianten. Aangezien iedere hybride automaat met elke gewenste nauwkeurigheid benaderd kan worden door een lineair hybride automaat, opent dit de weg om veel meer hybride systemen compositioneel te verifiëren door zowel componenten als specificatie te benaderen met lineair hybride automaten. Merk op dat onze definitie van onbeperkte invoer impliceert dat de invoer niet direct invloed kan uitoefenen op de geregelde variabelen, waardoor een dynamiek als $\dot{x} = u$ uitgesloten is. Deze automaten kunnen echter wel gebruikt worden om bemonsterde systemen met terugkoppeling te modelleren, aangezien voor deze systemen de waarde van de invoer alleen relevant is op discrete tijdstippen. Voor andere klassen dient bij compositioneel redeneren de TTS-semantiek te worden toegepast vóór de compositie-operator, die abstraheert van continue interactie tussen componenten. Dynamisch correspondeert dit met het aannemen van open invoer en kan het leiden tot een schending van nonconvexe invarianten. Zo betekent een differentiaalvergelijking $\dot{x} = u$ met open u dat $x(t)$ willekeurige waarden kan aannemen voor $t > 0$. Dit is een drastische overapproximatie die de compositionele analyse van bijvoorbeeld systemen met continue terugkoppeling verhindert. Deze overapproximatie kan tot op zekere hoogte worden ondervangen door aan te nemen dat de invoer of de afgeleide daarvan begrensd is (in het bijzonder bij aanname/garantie-redeneringen). Het gebruik van aanname/garantie-redeneringen voor hybride systemen lijkt zonder meer gerechtvaardigd en nuttig. Allereerst kan de abstractie van zelfs een enkele continue variabele de rekenkosten met diverse ordes van grootte reduceren. Ten tweede, aangezien een continue toestandsruimte een metriek heeft, kunnen eigenschappen zoals invarianten vaak compact gerepresenteerd worden. Dit is met name interessant indien men in staat is om een *nette* specificatie te geven, dat wil zeggen een specificatie die onafhankelijk is van de feitelijke implementatie.

Experimentele versies van algoritmen voor het checken van simulaties en aanname/garantie-redeneringen, alsmede voor het uitvoeren van bereikbaarheidsanalyses, zijn geïmplementeerd in een software tool genaamd PHAVer, dat vrij beschikbaar is [Fre05].

Zusammenfassung

Mit dem Einzug von Mikroprozessoren in nahezu alle elektrischen Apparate und technischen Prozesse spielen auch Software-Regelungen von Hardware-Systemen eine zunehmend wichtige Rolle. Die Interaktion von Software mit physikalischen Elementen und Prozessen kann ein komplexes, gemischt kontinuierlich-diskretes Verhalten hervorrufen, das sowohl die analytische Kapazität der klassischen, kontinuierlichen, Regelungstheorie als auch die der klassischen, diskreten, Informatik übersteigt. Solche Systeme werden als *hybrid* bezeichnet. Viele sicherheitskritischen Anwendungen, z.B. Flugsicherungen oder Drive-by-Wire-Systeme, weisen ein solches Verhalten auf. Es ist das Ziel der *formalen Verifikation*, d.h. einer mathematischen Modellierung des Systems und eines formellen Beweises, die Sicherheit eines Systems auf der Entwurfs-Ebene zu garantieren. Während formale Verifikation sich als Standardwerkzeug in der Entwicklung von diskreten Systemen etabliert hat, so z.B. bei digitalen Schaltkreisen, steht der Anwendung bei hybriden Systemen die inherente Komplexität kontinuierlich-diskreten Verhaltens im Weg. Verifikation ist in der Anwendbarkeit auf relativ einfache Systeme beschränkt, da der Berechnungsaufwand exponentiell mit der Anzahl der Komponenten und Variablen ansteigt. Dieses Phänomen wird als Zustandsexplosion bezeichnet. Bei hybriden Systemen ist sie besonders drastisch, da die Komplexität der Analyse kontinuierlicher Dynamik dramatisch von der Anzahl der Zustandsvariablen abhängt.

Um die Verifikation grösserer Systeme zu ermöglichen, erweitert diese Dissertation zwei fundamentale Ansätze aus dem Bereich der diskreten Systeme, Abstraktion und kompositionelle Beweise, auf den Bereich der hybriden Systeme aus. In einer *Abstraktion* eines Systems wird versucht, Information, die für den Beweis der Sicherheit unwesentlich sind, zu vernachlässigen, und so ein einfacheres Modell zu erhalten. Ein derart vereinfachtes Modell muss also konservativ in Bezug auf diese Eigenschaften sein. Eine *kompositionelle Analyse* untersucht Teilsysteme derart, dass daraus auf Eigenschaften des Gesamtsystems geschlossen werden kann. Eine besonders effektive Form der kompositionellen Analyse ist der *Anname/Garantie-Beweis*, in dem die Spezifikation eines Teilsystems als Annahme das Verhalten der anderen Teilsysteme berücksichtigt wird, und so zu einem zirkulären oder kettenartigen Beweis führt.

Um formal zu zeigen, dass ein Modell tatsächlich eine Abstraktion eines anderen ist, konstruiert man eine *Simulationsrelation* zwischen den Zuständen der Modelle. Ein Zustand *simuliert* einen anderen wenn von dort aus das gleiches, oder sogar mehr, Ver-

halten möglich ist. Intuitiv ist dies konservativ in Bezug auf Sicherheits-Eigenschaften, da Verhalten, das im abstrakten Modell nicht möglich ist, auch nicht im konkreten System auftreten kann. Ob eine kompositionelle Analyse korrekt ist, hängt davon ab, wie man Simulation definiert, und nicht alle Arten von Simulation lassen sich für hybride Systeme effizient berechnen. Das zentrale Ziel dieser Dissertation ist, Kompositionalität fuer eine einfach berechenbare Art von Simulation und für eine relevante Klasse von hybriden Systemen zu zeigen.

Die Arbeit gliedert sich in drei Teile: Im ersten Teil wird der Ansatz der kompositionellen Simulation für diskrete Systeme nach Grumberg und Long [GL91] verallgemeinert, und zusätzliche Beweisregeln fuer die kompositionelle Analyse aufgestellt. Im zweiten Teil wird gezeigt, dass sich dieser Ansatz direkt auf hybride Systeme anwenden lässt, wenn die Teilsysteme keine gemeinsamen Variablen haben. Als Semantik werden dazu beschriftete Transitionensysteme nach Henzinger eingesetzt [Hen96], die sogenannte TTS-Semantik. Im dritten Teil werden hybride Systeme mit gemeinsamen Variablen betrachtet. Dazu wird ein Modell für hybride Systeme mit Ein-/Ausgabe-Verhalten erarbeitet, und die Semantik geeignet modifiziert. Damit lässt sich zeigen, dass sich zwei relevante Klassen an hybriden Systemen kompositionell mit TTS-Semantik analysieren lassen. Bei den Klassen handelt es sich um hybride Systeme mit unbeschränkten Eingängen, und um hybride Systeme mit stückweise konstant begrenzten Ableitungen und linearen Beschränkungen. Letztere Klasse ist besonders relevant, da sich damit nach Henzinger et al. jedes hybride System beliebig genau konservativ approximieren lässt [HHWT98]. Somit öffnet diese Dissertation den Weg zu einer algorithmischen kompositionellen Analyse beliebiger hybrider Systeme.

Algorithmen für die Berechnung von Simulationsrelationen, kompositionellen Beweisen und fuer Erreichbarkeitsanalyse wurden in einem Werkzeug namens PHAVer implementiert, dass öffentlich verfügbar ist [Fre05]. PHAVer benutzt exakte Arithmetik, und kann damit Simulationsrelationen und Erreichbarkeit für Lineare Hybride Automaten exakt berechnen, und für hybride Systeme mit affiner Dynamik eine garantiert konservative Überapproximation erzeugen. Der Komplexitätszuwachs während der Analyse wird mittels Limitierung der Anzahl der Bits von Koeffizienten und der Anzahl der Beschränkungen in linearen Prädikaten kontrolliert. Dieser Ansatz zur Erreichbarkeitsanalyse hat sich an einem Benchmark-System selbst gegenüber nichtkonservativen Methoden als vorteilhaft erwiesen, und es ist damit gelungen, Systeme zu analysieren, die bislang von keinem anderen Werkzeug behandelt werden konnten.

Curriculum Vitae

Goran Frehse was born on January 31st, 1973, in Lindenberg im Allgäu, Germany. From 1992 to 1999 he studied Electrical Engineering and Information Technology at the Universität Karlsruhe (TH), Germany, majoring in control theory with a thesis on process control of hybrid systems. The academic year 1996/97 he studied at the Institut National des Sciences Appliquées de Lyon, co-authoring a thesis on fuzzy logic control of a welding process. In 2000 he became a PhD student at the Process Control Laboratory, Department of Biochemical and Chemical Engineering, Universität Dortmund, Germany, under the supervision of Prof. Sebastian Engell. During this time, he undertook study visits to Prof. Kim Guldstrand Larsen at the Distributed Systems and Semantics Unit of the Department of Computer Science at Aalborg University, Denmark, and to Prof. Frits W. Vaandrager at the Nijmegen Institute for Computing and Information Sciences, FNWI, Radboud University in Nijmegen. Since December 2003 he is a visiting researcher under Prof. Bruce H. Krogh at the Department of Electrical and Computer Engineering of Carnegie Mellon University in Pittsburgh, Pennsylvania.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division

of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty

of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-*. Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14