CARL VON OSSIETZKY universität OLDENBURG

BERICHTE

AUS DEM DEPARTMENT FÜR INFORMATIK
der Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften

Herausgeber: Die Professorinnen und Professoren des Departments für Informatik

Similarity, Logic, and Games
Bridging Modeling Layers of Hybrid Systems

Jan-David Quesel

Dissertation

Nummer 03-13 – Juli 2013
ISSN 1867-9218

CARL VON OSSIETZKY universität OLDENBURG

BERICHTE

AUS DEM DEPARTMENT FÜR INFORMATIK
der Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften

Herausgeber: Die Professorinnen und Professoren des Departments für Informatik

Similarity, Logic, and Games
Bridging Modeling Layers of Hybrid Systems

Jan-David Quesel

Dissertation

Nummer 03-13 – Juli 2013
ISSN 1867-9218

Gutachter: Prof. Dr. E.-R. Olderog
Prof. Dr.-Ing. J. Raisch (TU Berlin)

Datum der Einreichung: 03.04.2013
Datum der Verteidigung: 05.07.2013

Author's address:
Jan-David Quesel
Fakultät II, Department für Informatik
Abteilung „Entwicklung korrekter Systeme“
26111 Oldenburg
Germany

E-mail: Quesel@Informatik.Uni-Oldenburg.DE

CARL VON OSSIETZKY universität OLDENBURG
FAKULTÄT II
DEPARTMENT FÜR INFORMATIK

Carl von Ossietzky Universität Oldenburg
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Similarity, Logic, and Games

Bridging Modeling Layers of Hybrid Systems

Dissertation zur Erlangung des Grades eines Doktors der Naturwissenschaften

vorgelegt von

Jan-David Quesel

Oldenburg, April 3, 2013

Abstract

Specifications and implementations of complex physical systems tend to differ as low-level effects such as sampling are often ignored when high-level models are created. Thus, the low-level models are often not exact refinements of the high-level specification. However, intuitively we would consider them as similar. To bridge the gap between these models, we study notions of similarity and robust refinement relations for hybrid systems. We identify a family of such relations which permit certain bounded deviations in the behavior of a system specification and its implementation in both values of the system variables and timings. We show that for this relaxed version of refinement a broad class of properties is preserved. This includes stability, safety, as well as bounded response properties. The question whether two systems are in refinement relation can be reduced to a reachability problem for hybrid games.

For the study of parametric hybrid games, we propose a new logic, called *differential dynamic game logic* (dDG$\mathcal{L}$), and develop a theorem prover for it. We give an operational and a modal semantics of dDG$\mathcal{L}$ and prove their equivalence. To allow for deductive reasoning, we exploit the fact that dDG$\mathcal{L}$ is a conservative extension of differential dynamic logic (d$\mathcal{L}$). Subsequently, we provide rules for extending the d$\mathcal{L}$ sequent proof calculus to handle the dDG$\mathcal{L}$ specifics. Furthermore, we have implemented dDG$\mathcal{L}$ in our theorem prover KeYmaera. We demonstrate the strength of dDG$\mathcal{L}$ by applying KeYmaera to a case study in which a robot plays a game against other agents in a factory automation scenario.

KeYmaera is a theorem prover for hybrid system verification. It reduces the verification task to smaller subtasks that can be decided by quantifier elimination. Unfortunately, quantifier elimination over the reals is doubly exponential in the number of quantifier alternations already in theory and even in the number of variables in many implementations. Therefore, we compare different implementations of procedures for quantifier elimination and alternative methods for dealing with these subtasks.

We show that our dDG$\mathcal{L}$-based approach for proving that two hybrid systems are in robust refinement relation can be effectively used. For this, we present a case study from the domain of train control with a safe specification using instantaneous and imperfect implementations which suffer from communication delays.

Zusammenfassung

Bei der Entwicklung diskret-kontinuierlicher Systeme gibt es häufig Abweichungen zwischen der Spezifikation und der tatsächlichen Implementierung. Diese entstehen zum Beispiel durch maximale Abtastraten, die in der Spezifikation zu hoch angesetzt waren. Dadurch gibt es keine Verfeinerungsbeziehung im klassischen Sinne einer Mengeninklusion von Trajektorien zwischen den beiden Systemen (Spezifikation und Implementierung), allerdings kann man sie intuitiv durchaus als ähnlich bezeichnen.

In dieser Arbeit stellen wir einen Ähnlichkeitsbegriff vor, der es ermöglicht Eigenschaften, die für die Spezifikation gezeigt wurden, in abgeschwächter Form auf die Implementierung zu übertragen. Um zu zeigen, dass zwei Systeme in diesem Sinne ähnlich sind, konstruieren wir ein diskret-kontinuierliches Spiel, wobei die Existenz einer Gewinnstrategie für einen bestimmten Spieler die Ähnlichkeit der Systeme bezeugt. Zur Untersuchung dieser Spiele führen wir die Logik *Differential Dynamic Game Logic* (dDG$\mathcal{L}$) als konservative Erweiterung von Differential Dynamic Logic (d$\mathcal{L}$) ein. Darüber hinaus erweitern wir den Beweiskalkül für d$\mathcal{L}$ so, dass mit seiner Hilfe sich die semantische Eigenschaft der Ähnlichkeit auf syntaktische Umformungen von dDG$\mathcal{L}$-Formeln zurückführen lässt. Die Regeln des Kalküls zerlegen Beweisverpflichtungen in kleinere Teilziele. Hierbei beschreiben die Beweisregeln für die Modalitäten der Logik eine symbolische Ausführung der in den Modalitäten enthaltenen Programme. Das Ziel des Kalküls ist es, Beweisverpflichtungen in Prädikatenlogik über den reellen Zahlen zu erzeugen. Diese Logik ist entscheidbar, jedoch ist die Komplexität der Entscheidungsprozedur doppelt exponentiell in der Anzahl der Quantorenwechsel. Schlimmer noch, viele Implementierungen sind bereits doppelt exponentiell in der Anzahl der Variablen. Daher betrachten wir Alternativen für Teilfragmente dieser Logik, um die Beweisziele im Einzelfall schneller schließen zu können.

Den Beweiskalkül für dDG$\mathcal{L}$ haben wir in KeYmaera implementiert. Der Theorem-Beweiser KeYmaera dient zur Verifikation diskret-kontinuierlicher Systeme. Um die Anwendbarkeit unserer Logik dDG$\mathcal{L}$ zu demonstrieren, betrachten wir eine Fallstudie, in der sich ein Roboter durch eine Fabrik bewegen muss, ohne die Kooperation anderer Maschinen vorraussetzen zu können.

Des Weiteren betrachten wir das European Train Control System als Fallstudie um zu zeigen, dass unser dDG$\mathcal{L}$-basierter Ansatz zum Nachweis

von Ähnlichkeit diskret-kontinuierlicher Systeme effektiv durchführbar ist. In dieser Fallstudie zeigen wir, dass Implementierungen mit Kommunikationsverzögerungen ähnlich zu einer Spezifikation sind, die von instantaner Kommunikation ausgeht.

Acknowledgements

First of all I would like to thank my advisor Ernst-Rüdiger Olderog for providing a great working environment, numerous discussions on different topics, and teaching me valuable lessons about academia in general. Speaking of a great working environment I am grateful to my (former) colleagues (in alphabetical order): André Platzer, Björn Engelmann, Ingo Brückner, Johannes Faber, Mani Swaminathan, Martin Hilscher, Roland Meyer, Sibylle Fröschle, Stephanie Kemper, Sören Jeserich, Sven Linker, and Tim Strazny. Not part of our working group but also an important part of my environment was Hendrik Radke. For discussions on the topic of hybrid systems I would like to thank Andreas Eggers. Especially the discussions we had in the office of Sven and Martin will be something I am going to miss in the future.

While working on this thesis I had the chance to collaborate with several other researchers. Without the numerous discussions on notions of similarity with Ernst-Rüdiger Olderog, Martin Fränzle, and Werner Damm in our so called "Sonnenzimmer", the results in thesis would not have been possible. Furthermore, I would like to thank André Platzer for his interest in my work and his constant encouragement. I also enjoyed working with Philipp Rümmer on algebra related topics. The results of this collaboration can also be found in this thesis.

Furthermore, I would like to thank Jörg Raisch for inviting me to give a talk in his working group, reviewing this thesis, and introducing me to Anne-Kathrin Hess. This meeting led to an extremely fruitful discussion on the relation of approximations and notions of similarity for which I am grateful to Anne-Kathrin as well.

For pointing out the similarities of control engineering and game theory I am really grateful to Anders Ravn. For the help on algebra I would like to thank Wiland Schmale and Florian Heß.

Parts of the theories in this thesis have been implemented in our theorem prover KeYmaera, which is based on KeY. As everyone should know there is one person who knows every little detail on KeY and helped me out so many times. For this I am grateful to Richard Bubel. Furthermore, I would like to thank all of the KeY team for showing interest in our adventure of extending KeY to the hybrid domain. Meanwhile, KeYmaera has grown into a major tool itself. This also led to many interesting discussions with what I like to call the KeYmaera team. Especially, I would

like to mention here: André Platzer, Sarah Loos, Nikos Arechiga, Stefan Mitch, and Khalil Ghorbal. In addition, several students assisted me with programming KeYmaera over the years, and thus I would like to thank Timo Michelsen, Zacharias Mokom, Jianyu Bao, Sören Dierkes, and Miro El Seidi for their contributions.

For reading preliminary versions of this thesis I would like to thank: Sven Linker, Ernst-Rüdiger Olderog, Carsten Quesel, André Platzer, Hendrik Radke, Christin Quesel, Tim Strazny, Martin Hilscher, and Andrea Quesel-Bedrich. I would like to point out that Sven Linker read almost every single line of this thesis for which I am in debt to him.

During the years while I was working on this thesis my parents, Carsten Quesel and Iris Heyen, constantly encouraged me to keep on going for which I am really grateful now.

Last but not least I would like to express my gratitude to my wife Christin Quesel for her constant support. Without her this thesis would not have been possible. Thank you.

From all the things I've lost, I miss my mind the most.
— Mark Twain

Contents

List of Figures

List of Tables

CHAPTER ONE

Introduction

In the beginning there was nothing, which exploded.
— Terry Pratchett

Contents

1.1 Motivation

Computers play an important role in our modern society. Office PCs, notebooks, tablets, and smartphones are the most prominent examples. However, computers are also used in many medical devices like cardiac

stimulators or infusion pumps, means of transportation like trains, cars, or airplanes, and power plants to control critical parts of the system.

Hence verification of complex physical systems is becoming more and more important. The development of those systems, however, follows a paradigm where, if formal specifications are used, relations to the actual implementations are often not formally proven.

In this work, we bridge this gap by providing a mathematical notion that connects specifications and their implementations. We therefore study directed quantitative notions of similarity and refinement. This is, we define a distance measure on the trajectories of the systems. These distances can then be used to compute how much the behavior of an implementation deviates from its specification. We then use these bounds in order to infer what properties are shared between the two systems.

"Safe by design" is a principle that among other notions inspired that of refinement. The idea of refinement is to start with a broad nondeterministic specification and add details one by one. Finally, one arrives at a deterministic implementation that covers all details including properties of the target hardware like sampling. The steps from specification to implementation are called refinement steps if they preserve the "behavior" of the system. That is, a specification A is a refinement of another specification B if, and only if, all the "behavior" we can observe of A could also be seen while observing B. This process guarantees that if B only shows "good behavior" then so does A.

Even though refinement is a classical engineering method, it often is not suitable in practice. What happens if some details only come up late in the design process? In order to stick to the paradigm just discussed it would be necessary to propagate these from a level closer to the implementation upwards to the specification (possibly by adding more nondeterminism). Afterwards, one needs to prove once more that the system behaves in a safe way. However, it could be the case that the methods used for proving specifications safe cannot deal with the level of detail or the amount of nondeterminism necessary. Thus in practice, often specifications do not take into account details like sampling. It might even be the case that the actual sampling rate is determined based on costs by some client. In order to allow for delaying such choices we study notions of relaxed or robust refinement that permit some deviations in the valuations of system variables as well as timings of events. That way, if our initial specification satisfies a strong property then a weaker version of this property can be transferred to its robust refinements.

Notions of similarity have been studied by physicists for over a hundred years. However, in physics the idea is to capture similar processes within a uniform mathematical model. This is important in order to extrapolate mathematical laws from a series of experiments [Web40]. If every experiment that deviates slightly in the inputs would result in totally different observed behavior of the system, then no such mathematical law could be formulated.

In the design process of ships, cars, and airplanes experiments on models are preformed frequently [Web40]. Engineers usually consider models that are geometrically similar in the sense that the model is a scaled version of the original system. In addition, the ratios of forces applied to the model and to the original system have to be constant. If that is the case then results obtained from the model experiment can be assumed to hold for the real system. This idea was generalized in the setting of fluid dynamics. Here, formalisms exist that allow for small deviations in the system values. Reynolds [Rey83] observed that as long as the quotient of kinetic energy of a fluid and energy loss due to friction of the object under observation in this liquid is approximately equal then the fluid flow along these objects will be almost the same.

Instead of conducting experiments, we study mathematical notions of similarity on models of systems and what this means for their observable behavior on a semantic level. In order to formalize observable behavior we consider logical formulas featuring modalities that allow for expressing temporal and real-time behavior. We thus generalize the ideas of Reynolds in the sense that we do not restrict ourselves to a specific application domain but instead cover arbitrary physical systems with computerized controllers. In the early 1990s, Anil Nerode suggested a name for this type of systems that would stick [Rav13]. He called them *hybrid systems*.

1.2 Real-Time and Hybrid Systems

A famous result of landmark importance for the study of computerized systems which control processes in the physical world was the introduction of *timed automata* by Alur and Dill [AD91] in 1991. Timed automata are finite automata extended with so called clocks. These are variables that evolve with a constant slope of 1 while the automaton is in some location. Transitions between these locations happen instantaneously. They might be guarded by some conjunction of restricted linear expressions over clocks.

In addition, they may reset a subset of the clocks to 0. Up until that point most formalisms were used only to express qualitative relations between events. To the best of our knowledge timed automata were the first model that allowed to express the time taken on a complete path of the automaton. In previous formalisms it was only possible to express that individual transitions happen within certain time bounds. Alur and Dill [AD91] further presented a PSPACE algorithm for deciding reachability of locations in timed automata. This rather simple model already allows for proving a variety of properties of different system specifications, and tools like UPPAAL [BLL+95], based on these results, are successfully applied in the industrial practice.

In 1992, on the *Hybrid Systems* workshop Alur, Courcoubetis, Henzinger, and Ho [ACHH92] and independently Nicollin, Olivero, Sifakis, and Yovine [NOSY92] proposed hybrid automata as an extension to timed automata. The most well-known reference on hybrid automata was written a few years later in 1996 by Henzinger [Hen96].

Unlike timed automata, hybrid automata are not restricted to variables acting as clocks. Instead, they allow for evolutions of the variables along solutions of differential equations. Furthermore, they relax the restrictions on transitions guards and resets. That is, they allow for arbitrary arithmetic expressions over system variables as transition guards and resets to arbitrary values also described by arithmetic expressions over the system variables. Unfortunately, this extension does not preserve the decidability result regarding reachability of locations. Already, for very restricted subsets of hybrid automata, so called *stopwatch automata*, i.e., timed automata that are allowed to stop the evolution of a single clock, reachability is undecidable [HKPV95]. Still algorithms where developed and implemented in various tools in order to check properties of these systems. One of the first such tools was HyTech [HHWT97], a model checker for linear hybrid automata.

1.3 Real Arithmetic

When checking properties of hybrid systems it is important to represent the reachable state space, i.e., the set of variable valuations that can be reached during an evolution of the system. As the variables are evaluated to real numbers the theory of real arithmetic is important. When dealing with a linear hybrid automaton the reachable state space can usually be

described sufficiently well by linear constraints for which fast computation algorithms exist. When it comes to more complex dynamics however, linear constraints only provide very coarse over-approximations of what is actually happening. Thus, Tarski's famous result [Tar51] that first-order logic over polynomial expressions with rational coefficients and real valuations of the variables is decidable comes in handy. However, the non-elementary complexity of his procedure makes it impractical. Fortunately, Collins discovered a faster method for this task [Col75]. Still, Collin's method is doubly exponential in the number of variables. Even worse, Davenport and Heintz [DH88] showed that any procedure for this problem is at least doubly exponential in the number of quantifier alternations. Still, many implementations of quantifier elimination procedures perform quite well and for the universal fragment even faster methods are available.

1.4 Modal and Dynamic Logics

For many applications, first-order logic provides a powerful tool for describing properties of systems. The bases of first-order logic is formed by Boolean connectives. These are motivated by the connectives used in our natural language. Therefore, the connectives are "and", "or", "if... then", and "if and only if". In addition the logic allows to express properties by stating the opposite, i.e., the use of negation. First-order logic further features quantification over objects.

In the beginning of the 20th century a discussion started about propositional logic and especially the implication operator. The implication operator seamed to suggest a connection between its premise and conclusion. However, if the conclusion is valid then any premise—related or not—will result in a valid formula. This bothered philosophers and resulted in an extension to propositional logic, the so called modal logics [Lew18]. A prominent example of a modal logic is intuitionistic logic [Bro07]. In intuitionistic logic only those statements hold, that can be constructively proven. This demands to determine the truth value of a statement either one has to constructively prove that it is true or constructively prove that its negation is true. If neither proof is known the truth value of the statement itself is unknown. Orlov [Orl28] and Gödel [Göd33] interpreted this logic as a modal logic with a modality that expresses "it can be constructively proven that". Other examples include multiple modalities like "it is possible that" and "it is necessary that". Prior [Pri55] studied temporal interpretation of these

modalities, i.e., modalities that express that something is "always" the case and its dual, i.e., something is "eventually" the case. He also introduced a binary temporal modality "until" and studied its relation to the unary modalities "always" and "eventually" [Pri67]. Pnueli [Pnu77] applied the idea of temporal logic to computer science problems thereby pioneering an important field of research. Temporal logic provides a qualitative notion of time in the sense that the order of events is expressible. However, it is not possible to express quantitative relations. Koymans' Metric temporal logic (MTL) [Koy90, OW08] is tailored to bridge this gap. MTL features for annotated modalities with intervals to qualify properties by modalities like "eventually with 2 time units". Of course combinations of first-order and modal logic have been considered. For an overview on first-order modal logic see [FM99].

Pratt [Pra76] studied how operational models of programs and modal logics could be connected. The resulting logic was *dynamic logic* in which statements like "for every execution of the program A it holds that" or "there is a execution of program A such that" can be made. Later Harel [HKT00] combined dynamic logic and first-order logic into a *first-order dynamic logic*.

Platzer [Pla10b] lifted Harel's ideas to the verification of hybrid systems and created a logic called *differential dynamic logic* $\mathsf{d}\mathcal{L}$ in which modalities of the form "for every run of the hybrid system A it holds that" and "there is a run of the hybrid system A such that" exist.

In this thesis, we will use metric versions of temporal logic as well as dynamic logics to formulate properties of hybrid systems.

1.5 Contributions

We present a family of robust refinement relations for hybrid systems. We show that stability properties can be transferred from a system to its robust refinements in a relaxed form. Furthermore, we present a variant of metric temporal logic (MTL), natural logic ($\mathcal{L}\natural$), together with a syntactic transformation function to compute for a given property how it is preserved under robust refinement.

We present a novel conservative extension to differential dynamic logic that can be used to express properties of hybrid games. The resulting logic is called *differential dynamic game logic* ($\mathsf{dDG}\mathcal{L}$). In order to clarify the relation to hybrid games, we present two semantics—a classical modal

semantics and a game semantics—and show that they are equivalent. Then we present a sequent proof calculus for dDG$\mathcal{L}$ and perform a case study.

We present two related approaches to show that a system is robustly refined by another one. The first one is based on a construction on hybrid automaton. The second approach is based on our new logic dDG$\mathcal{L}$.

The proof calculus for dDG$\mathcal{L}$ relies on a methods for dealing with real the resulting proof obligations in first-order logic over the reals. Therefore, we survey different methods for dealing with these tasks and present a novel method combining semidefinite programming for the Real Nullstellensatz and Gröbner bases that is complete for the universal fragment of real closed-fields and provides checkable certificates. Subsequently, we perform an experimental evaluation of the different methods on a large set of benchmarks which shows that our new methods outperforms many of the other approaches.

The *European Train Control System* (ETCS) serves as a case study to demonstrate the interplay of our methods. We show that safety properties of a specification modeling a train controller and radio block controller with instantaneous communications can be transferred to implementations with communication delays.

1.6 Structure of this Thesis

In Chapter 2 we recapitulate some basic definitions. We discuss notions of similarity and robust refinement in Chapter 3. Chapter 4 considers hybrid games as extensions to hybrid systems and presents an extension of differential dynamic logic [Pla08]. We call the resulting logic differential dynamic game logic (dDG$\mathcal{L}$). Chapter 5 relates robust refinement of hybrid systems and hybrid games. Subsequently, we discuss an implementation of the proof calculus and ways of dealing with the resulting arithmetical verification tasks in Chapter 6. As a showcase for the methods developed in the Chapters 3–5 we present a case study in Chapter 7 from the domain of train control. We summarize our results in Chapter 8 and sketch various directions for future work.

1.7 Sources

This thesis contains parts that were joint work with other authors and have been partially published before. Chapter 3 is a substantially extended version of joint work with Martin Fränzle and Werner Damm [QFD11]. Results from the same publication form the basis of Section 5.1. Chapter 4 contains results that were obtained as joint work with André Platzer and have been previously published in [QP12a]. The results in Chapter 6 were joint work with André Platzer and Philipp Rümmer [PQ08a, PQR09a]. We base our case study in Chapter 7 on joint work with André Platzer [PQ08b, PQ09a, PQ09b].

1.8 How to Read this Thesis

There are different dependencies between the chapters in this thesis. Every reader should have a look at Section 2.1 to get an overview of notions and abbreviations used throughout this thesis. Readers that are familiar with hybrid systems can skip Section 2.2 and refer back to the detailed definitions of syntax and semantics of hybrid programs (Section 2.2.2) and hybrid automata (Section 2.2.4) when needed.

For readers interested in our notions of robust refinement and our approach on proving that two systems are in refinement relation we suggest reading Chapter 3 and Section 5.1. Readers that want to go into more detail on that topic could then read up on differential dynamic game logic in Section 4.1 and Section 4.2 and our more general way of proving refinement relations using dDG$\mathcal{L}$ in Section 5.2.

For readers interested in our theorem prover KeYmaera, we suggest reading Section 2.2.2 about hybrid programs, Chapter 4 about differential dynamic game logic and especially our sequent proof calculus in Section 4.3, and Chapter 6 about implementation details with a special focus on dealing with the resulting verification conditions formulated in first-order logic over the reals.

CHAPTER TWO

Preliminaries

"There's a door"
"Where does it go?"
"It stays where it is, I think."
— Terry Pratchett

Contents

In this chapter we define basic terms and notions which will provide the formal basis for the next chapters. After fixing some notions and abbreviations in Section 2.1, Section 2.2 gives a brief introduction into hybrid systems and hybrid system modeling. For this purpose we will cover hybrid automata [ACHH92, NOSY92, Hen96] and hybrid programs [Pla10b] as operational models for hybrid systems. Subsequently, in Section 2.3 we discuss different logics to get a mathematical grip on formulating properties of such systems. The logics covered in this section range from propositional logic over first-order logic to differential dynamic logic ($\mathsf{d}\mathcal{L}$) [Pla10b]. The logic $\mathsf{d}\mathcal{L}$ is specifically tailored for hybrid system verification. In addition, we present an overview on metrics and norms in order to later measure distances between system trajectories in Section 2.4.

2.1 Notations and Abbreviations

First, let us fix some notations. This small section should serve as a reference for the notations. Most of these are repeated again where they are first needed.

We denote the set of natural numbers $\{0, 1, 2, \dots\}$ by $\mathbb{N}$, the set of rational numbers by $\mathbb{Q}$, and the set of real numbers by $\mathbb{R}$. The non-negative real numbers are denoted by $\mathbb{R}_{\geq 0}$. For a number r, we use $|r|$ to denote its absolute value whereas for a set X we use $|X|$ to denote its cardinality. For two sets X and Y we denote by $X \dot\cup Y$ their disjoint union. Functions denoted by $||\cdot||$ are assumed to be arbitrary norms if not mentioned otherwise. We denote by dom f the domain of a function f. Furthermore, we denote by $B_r \hat{=} \{x \mid x \in \mathbb{R}^n \wedge ||x|| < r\}$ the open n-dimensional ball with radius r centered at the origin.

We assume that hybrid programs are built using a finite, ordered set of variables V. States are mappings between these variables and real numbers. For ease of presentation we convert between states $\nu : Sta(V) \to \mathbb{R}$ and vectors $\nu \in \mathbb{R}^n$ where $n = |V|$ is implicitly assumed. This transformation is assumed to be uniquely defined by the order of the variables.

Throughout this thesis we use $\dot{x}$ to denote the time derivative of some variable x.

2.2 Models for Hybrid Systems

As this work focuses on mathematical models for physical systems, so called *hybrid systems* [Pla10b, ACHH92, NOSY92, Hen96], we will, in this chapter, present different representation of these models. First, we will give an introduction to what hybrid systems are. Then, we define a program notation for hybrid systems, so called *hybrid programs* [Pla10b]. Subsequently, we define a notion of *hybrid automata* [ACHH92, NOSY92, Hen96], which could be seen as the standard model for hybrid systems.

2.2.1 Hybrid Systems by Example

Hybrid systems occur in many places in our day to day life. Examples include trains [PQ08b, PQ09a], cars [ALPK12], planes [Pla10b] but also drug infusion pumps [SHL11], electric heaters [HHWT97], and many more. To pick an example we focus on trains in this section. The movement of trains can be described by differential equations. Kinematic models based on Newton's laws of mechanics are sufficient for train interactions where p is the position of the train, v its velocity and a its acceleration. All these state variables are functions in time t.

$$\frac{\mathsf{d}p}{\mathsf{d}t} = v, \frac{\mathsf{d}v}{\mathsf{d}t} = a \tag{2.1}$$

As time domain we use the non-negative real numbers, denoted by $\mathbb{R}_{\geq 0}$, and instead of $\frac{\mathsf{d}p}{\mathsf{d}t}$ we write $\dot{p}$ for the time-derivative of p. Equation (2.1) specifies that the position p changes over time according to the value of the velocity. The velocity itself changes according to the acceleration a. However, the equation does not specify how the acceleration a evolves. To make the model of the dynamics more realistic we could add another differential equation $\dot{a} = j$ where j is the jerk, but then the question just shifts over to j as it is not described how j evolves. We follow the *explicit change* principle. That is, no variable changes unless the model explicitly specifies how it changes. In particular, the absence of a differential equation for a in (2.1) indicates a is constant during the continuous evolution.

If we want to model an *analog controller* for a, we can replace a in (2.1) by a term that describes how the analog controller sets the acceleration a, depending on the current position p and velocity v. For example, if d is the set-value for the velocity, we describe a simple proportional controller with gain K_p by the differential equation $\dot{p} = v, \dot{v} = K_p(v - d)$.

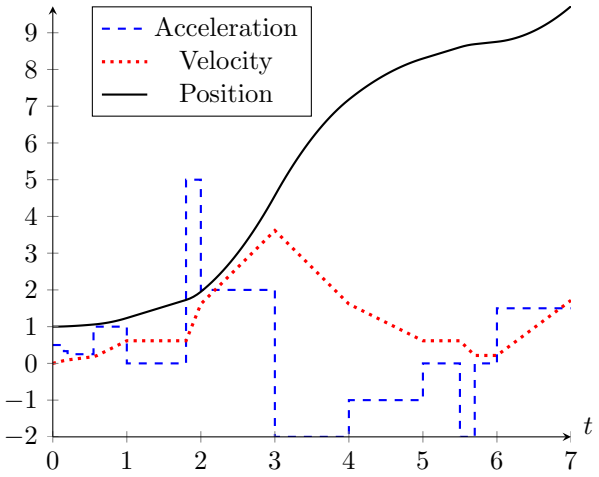


Figure 2.1: One example trace of a hybrid system for train dynamics (2.1) with the acceleration signal changing as indicated over time

A common alternative is to use a *digital controller*, which turns the purely continuous dynamical systems into a *hybrid system* that exhibits both discrete and continuous dynamics. An example trajectory of the train dynamics of (2.1), controlled by a discrete controller for the acceleration a that changes its values at various instants in time is shown in Figure 2.1.

The figure traces the values of the system state variables p, v, and a over time $t \in \mathbb{R}_{\geq 0}$. The acceleration a changes its value instantaneously according to some discrete controller (not specified in (2.1)) and this effect propagates to the velocity and position according to the relations given by the differential equation (2.1).

Given a target speed r we may want to build a digital controller that chooses a constant positive acceleration of A if the current speed is too low and a constant deceleration of $-b$ if it is too high. We specify the controller by the following program:

$$(\text{if } v \leq r \text{ then } a := A \text{ else } a := -b \text{ fi}; (\dot{p} = v, \dot{v} = a))^* \tag{2.2}$$

The *if-fi* statement is a case distinction. It first checks whether $v \leq r$ holds. If that is the case then $a := A$ is executed. This means that the value of

a gets updated to the value of A, which we could think of as the maximal acceleration. Otherwise, i.e., if $v > r$ then the assignment $a := -b$ gets executed, assigning the maximal deceleration of $-b$ to a. The operator ; is for sequential composition. That is, after the first statement finishes (here, the if statement) the next statement is executed ($\dot{p} = v, \dot{v} = a$). Hence, after the controller chooses an acceleration, the variables evolve according to the solution of this differential equation system for some time $t \in \mathbb{R}_{\geq 0}$. This evolution time is chosen nondeterministically. During this evolution the acceleration a is constant. However, the position and velocity change continuously in parallel. Operator * denotes nondeterministic repetition like in regular expressions. That is, operator * repeats arbitrarily often. The loop enables the discrete controller to update the acceleration.

A common and useful assumption when working with hybrid systems is that discrete actions do not consume time (whenever they do consume time, it is easy to transform the model to reflect this just by adding explicit extra delays). Because discrete actions are assumed not to consume time, multiple discrete actions can occur at the same real point in time. To reflect this, we model the time as a sequence of real-valued functions. Hence the time model for hybrid systems, called *hybrid time*, is given by $\mathbb{N} \times \mathbb{R}_{\geq 0}$.

The program (2.2) does not specify when the continuous evolution stops to give the discrete controller another chance to react. Thus, we add a clock variable c to model a time-triggered architecture.

$$(\text{if } v \leq r \text{ then } a := A \text{ else } a := -b \text{ fi}; c := 0; (\dot{p} = v, \dot{v} = a, \dot{c} = 1 \ \& \ c \leq \varepsilon))^* \tag{2.3}$$

The clock c is reset to zero by the discrete assignment $c := 0$ before every continuous evolution and then evolves with a constant slope of $\dot{c} = 1$. Its value is bounded by a constant symbol ε. Therefore, the variables of the system in (2.3) evolve for at most ε time units, i.e., the discrete controller is invoked at least every ε time units. For hybrid systems it is often important that the set of reachable states is restricted. This restriction is usually called a safety property. For example, a safety property for our train could be that it should not considerably exceed the speed r. The model paradigm, used in the last example, where we invoke the controller not at exact sampling times but at least every ε time units, ensures that if the controller is implemented on faster hardware, then it will still have the same safety properties. In Figure 2.1 the acceleration is changed for example almost every 1 time unit. Still, there are, in total, more changes to the acceleration. The formula $c \leq \varepsilon$ that is separated from the differential

equation by the symbol & is an *evolution domain constraint*. Evolution domain constraints are formulas that restrict the continuous evolution of the system to stay within that domain. This is, the continuous evolution starts within the specified domain and must stop before it leaves this region. Note that the model (2.3) only puts an upper bound on the duration of a continuous evolution, not a lower bound. The discrete controller can react faster than ε and, in fact, in Figure 2.1, it does react more often.

The next extension to our model adds nondeterminism to the choice of the acceleration. If we replace the assignment $a := A$ by $a := A \cup a := 0$ (read "a becomes A *or* a becomes 0), then the controller can always choose to keep its current velocity instead of accelerating further. We use $\cup$ to be a nondeterministic choice. Hence the program can unconditionally follow either way.

$$(\text{if } v \leq r \text{ then } a := A \cup a := 0 \text{ else } a := -b \text{ fi}; \\ c := 0; (\dot{p} = v, \dot{v} = a, \dot{c} = 1 \& c \leq \varepsilon))^* \quad (2.4)$$

2.2.2 Hybrid Programs

The program model for hybrid systems that we have illustrated by example is called *hybrid programs* (HP) [Pla08,Pla10a,Pla10b]. The *syntax of hybrid programs* is shown together with an informal semantics in Tabular 2.1. The basic terms (called θ in the table) are either rational number constants, real-valued variables or *polynomial arithmetic expressions* built from those. That is, they are generated from the grammar

$$\theta ::= q \mid v \mid \theta + \theta \mid \theta \cdot \theta$$

where $q \in \mathbb{Q}$ are rational numbers and $v \in V$ are variables. Observe that, these terms can be nonlinear as they are closed under multiplication. Further, we allow fractions of rational polynomials if it is clear from the context that they are defined.

The effect of $x := \theta$ is an instantaneous discrete jump assigning the value of θ to the variable x. For example in Figure 2.1, the acceleration a changes instantaneously at time 1.8 from 0 to 5, by the discrete jump $a := A$ for A having value 5. For a train with current velocity v the deceleration necessary to come to a stop within distance m is given by $-\frac{v^2}{2m}$. The controller could assign this value to the acceleration by the

Table 2.1: Statements of hybrid programs (F is a first-order formula, α, β are hybrid programs)

Statement	Effect
$\alpha;\ \beta$	*sequential composition* where β starts after α finishes
$\alpha\ \cup\ \beta$	*nondeterministic choice*, following either alternative α or β
α^*	*nondeterministic repetition*, repeating α n times for any $n \in \mathbb{N}$
$x_1 := \theta_1, \ldots, x_n := \theta_n$	*parallel discrete assignment* of the values of terms θ_i to the variables x_i (*parallel jump*)
$x := *$	*nondeterministic assignment* of an arbitrary real number to x
$(\dot{x_1} \sim_1 \theta_1, \ldots, \dot{x_n} \sim_n \theta_n \& F)$	*continuous evolution* of x_i along differential (in)equation system $\dot{x_i} \sim_i \theta_i$, with $\sim_i \in \{\leq, =\}$, restricted to evolution domain F
$?F$	test if formula F holds at current state, abort otherwise
if(F) then α	perform α if F is true at current state, do nothing otherwise
if(F) then α else β	perform α if F is true at current state, perform β otherwise

assignment $a := \frac{v^2}{2m}$. In addition, we allow parallel assignments to multiple variables. For example, $x := y, y := x$ switches the values of x and y.

The effect of $\dot{x} = \theta \,\&\, F$ is an ongoing continuous evolution controlled by the differential equation $\dot{x} = \theta$ that is restricted to remain within the evolution domain F, which is a formula of arithmetic. The evolution is allowed to stop at any point in F but it must not leave F. Systems of differential equations are defined accordingly: $\dot{p} = v, \dot{v} = -b \,\&\, v \geq 0$, for instance, characterizes the braking mode of a train with braking force b that holds within $v \geq 0$ and stops any time before $v < 0$. The extension to systems of differential equations is straight forward, see [Pla08, Pla10a, Pla10b]. The language supports higher-order derivatives by using auxiliary variables. That is, in the previous example the second derivative of the position is the acceleration, i.e., $\ddot{p} = -b$.

For discrete control, the test action $?F$ is used to define conditions. It succeeds without changing the state if F is true in the current state, otherwise it aborts all further evolution. For example, a train controller can check whether the chosen acceleration is within physical limits by

$$?-b \leq a \leq A \ .$$

If a computation branch does not satisfy this condition, the branch is discontinued and aborts.

From these basic constructs, more complex hybrid programs can be built similar to regular expressions. The *sequential composition* $\alpha;\beta$ expresses

that hybrid program β starts after hybrid program α finishes, as in Expression (2.2). The nondeterministic choice $\alpha \cup \beta$ expresses alternatives in the behavior of the hybrid system. Nondeterministic repetition α^* says that the hybrid program α repeats an arbitrary number of times, including zero. These operations can be combined to form any other control structure.

For instance, $(?v \geq r; a := A) \cup (?v \leq r; a := -b)$ says that, depending on the relation of the current speed v of some train and a given target speed r, a is chosen to be the maximum acceleration A if $v \leq r$ or maximum deceleration $-b$ if $v \geq r$. If both conditions are true (hence, $v = r$) the system chooses either way. Note that the choice between the two branches is made nondeterministically. However, the test statements abort the program execution if the left branch was chosen in a state where $v \geq r$ does not hold, or the right branch was chosen in a state where $v \geq r$ was not satisfied. As abbreviations, we add *if-statements* to our program syntax with the usual meanings from programming languages. The if-statement can be expressed using the test action, sequential composition and the choice operator.

$$\begin{aligned} \text{if } F \text{ then } \alpha \text{ fi } &\equiv (?F; \alpha) \cup (?\neg F) \\ \text{if } F \text{ then } \alpha \text{ else } \beta \text{ fi } &\equiv (?F; \alpha) \cup (?\neg F; \beta) \end{aligned}$$

The semantics of the first variant is as follows: If condition F is true, the then-part α is executed otherwise the statement has no effect. For the second statement the semantics is that if condition F is true, the then-part α is executed otherwise the else-part β is performed. Note that, even though we use nondeterministic choice in the encoding, the choice becomes deterministic as the conditions in the test actions are complementary, so exactly one of the two tests $?F$ and $?\neg F$ fails in any state.

When combining choices and test it is important to make sure that the model does not get blocked in an unnatural way. For example, the program

$$(?v < 3; \dot{v} = A) \cup (?v > 5; \dot{v} = -b)$$

cannot evolve if v is between 3 and 5. Therefore, it is good modeling practice to have at least one branch for each case. Evolution domain constraints also need to be designed with care. For example, the hybrid program

$$((\dot{v} = -b \& v \geq 0) \cup (\dot{v} = -b \& v < 0))^*$$

has disjoint evolution domain constraints. When $v = 0$, the system cannot switch to the second choice, because its evolution constraint $v < 0$ is not

satisfied for the initial state. Therefore, the evolution domains have to be overlapping if switching between the differential equations is desired behavior.

The *nondeterministic assignment* $x := *$ assigns any real value to x, thereby expressing unbounded nondeterminism, e.g., in choices for controller reactions. For instance, the idiom $a := *; ?a > 0$ nondeterministically assigns any positive value to the acceleration a, because only positive choices for the value of a will pass the subsequent test $?a > 0$.

We now define the semantics of hybrid programs formally in terms of hybrid traces like in [Pla07c]. Let V be a finite set of variable names. A *state* is a map $\nu : V \to \mathbb{R}$. Additionally, Λ is a failure state that is reached iff a test during a program execution failed. The set of all states over V is denoted as $\mathrm{Sta}(V)$. Let $\nu(\theta)$ denote the value of a term θ in a state ν and $\nu[x \mapsto d]$ denote the state that coincides with ν up to the valuation of x which is changed to the value of d.

Definition 1 (Hybrid Trace [Pla07c])**.** *A* hybrid trace *is a (nonempty) finite or infinite sequence $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$ of flows with their respective durations $r_i \in \mathbb{R}$ (for $i \in \mathbb{N}$), i.e., functions $\sigma_i : [0, r_i] \to Sta(V)$. For a state $\nu \in Sta(V)$, $\hat{\nu} : 0 \mapsto \nu$ is the* point flow *at ν with duration 0. A trace* terminates *if it is a finite sequence $\sigma = (\sigma_0, \sigma_1, \dots, \sigma_n)$ and $\sigma_n(r_n) \neq \Lambda$. In that case, the* last state *$\sigma_n(r_n)$ is denoted by $last(\sigma)$. The* first state *$\sigma_0(0)$ is denoted by $first(\sigma)$. The* composition *of two traces $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$, $\varsigma = (\varsigma_0, \varsigma_1, \varsigma_2, \dots)$ is defined by*

$$\sigma \circ \varsigma = \begin{cases} (\sigma_0, \dots, \sigma_n, \varsigma_0, \varsigma_1, \dots) & \text{if } \sigma \text{ terminates and } last(\sigma) = first(\varsigma) \\ \sigma & \text{if } \sigma \text{ does not terminate} \\ \text{not defined} & \text{otherwise .} \end{cases}$$

Further, we denote the set of all traces *by $\mathfrak{T}$.*

Note that the tuple $(\hat{\nu}, \hat{\omega})$ denotes a hybrid trace with length 0 consisting of two point flows. These flows occur in the order first $\hat{\nu}$ and then $\hat{\omega}$ at the same real-valued point in time.

Definition 2 (Trace semantics of hybrid programs [Pla07c])**.** *The* trace semantics *$\tau(\alpha)$ of a hybrid program α, is the set of all its possible hybrid traces and is defined inductively as follows:*

1. $\tau(x_1 := \theta_1, \ldots, x_n := \theta_n) =$

$$\{(\hat{\nu}, \hat{\omega}) \mid \omega = \nu[x_1 \mapsto \nu(\theta_1)] \ldots [x_n \mapsto \nu(\theta_n)], \nu \in Sta(V)\}$$

2. $\tau(x := *) = \{(\hat{\nu}, \hat{\omega}) \mid \omega = \nu[x \mapsto \nu(r)], r \in \mathbb{R}, \nu \in Sta(V)\}$

3. $\tau(\dot{x} = \theta \& \chi) = \{(f) \mid 0 \leq r \in \mathbb{R}$ *and* $f : [0, r] \to Sta(V)$ *is such that the function* $f(\zeta)(x)$ *is continuous in* ζ *on* $[0, r]$ *and has a derivative of value* $f(\zeta)(\theta)$ *at each* $\zeta \in (0, r)$*, while* $f(\zeta)(y)$ *is constant for each variable* y *without a differential equation. Also each intermediate value satisfies the invariant* χ*.*$\}$

4. $\tau(?\chi) = \{(\hat{\nu}) \mid \nu \models \chi\} \cup \{(\hat{\nu}, \hat{\Lambda}) \mid \nu \not\models \chi\}$

5. $\tau(\alpha \cup \beta) = \tau(\alpha) \cup \tau(\beta)$

6. $\tau(\alpha; \beta) = \{\sigma \circ \varsigma \mid \sigma \in \tau(\alpha), \varsigma \in \tau(\beta)$ *if* $\sigma \circ \varsigma$ *is defined*$\}$

7. $\tau(\alpha^*) = \bigcup_{n \in \mathbb{N}} \tau(\alpha^n)$*, where* $\alpha^{n+1} = \alpha^n; \alpha$ *for* $n \geq 1$*,* $\alpha^0 = ?true$*.*

Here, $\models$ denotes the consequence relation for first-order formulas over the reals. We will formally define this relation in the following sections. Later, we will allow formulas from more complex logics in places where this definition is restricted to first-order. Observe that, in the definition of the semantics of continuous evolutions, the function f projected to the component x describes the solution to the differential equation. The restriction that each intermediate value has to satisfy χ ensures that the evolution does not leave its evolution domain region described by χ.

Let $\rho(\alpha)$ denote the relation between all states that are connected by a terminating trace of α, i.e.,

$$\rho(\alpha) = \{(\nu, \omega) \mid \exists(\sigma_0, \ldots, \sigma_n) \in \tau(\alpha) : \sigma_0(0) = \nu \wedge \sigma_n(r_n) = \omega \neq \Lambda\}$$

where $r_n = \max(\text{dom } \sigma_n)$.

2.2.3 Differential-algebraic Constraints

For modeling more complex dynamics, we use so called *differential-algebraic constraints* [Pla10a, Pla10b], i.e., first-order arithmetic constraints whose free variables are a subset of the system state variables and their derivatives. For example, let the position of a train with longitudinal and lateral

dynamics be denoted by (x, y). Then the differential-algebraic constraint $\dot{x} + \dot{y} \leq 1$ gives a constraint that limits the speed of the train to a maximum of 1. For our one dimensional train example, the differential-algebraic constraint $\exists d\, \dot{p} = v \wedge \dot{v} = a + d \wedge d_{min} \leq d \leq d_{max}$ models a continuous disturbance to the choice of the acceleration in the interval $[d_{min}, d_{max}]$. For these *differential-algebraic constraints*, the system follows a trajectory satisfying the constraint locally, see [Pla10a, Pla10b] for details. If there is no such trajectory, the execution of the program fails.

2.2.4 Hybrid Automata

Of course hybrid programs are not the only formalism to model hybrid systems. The most common formalism to specify hybrid systems are hybrid automata [ACHH92, NOSY92, Hen96].

Definition 3. *A* hybrid automaton *is a tuple* $H = (U, X, L, E, F, Inv, Init)$ *where*

- $V := U \dot{\cup} X$ *is a set of real-valued variables where* U *is the set of external variables and* X *contains the internal ones.*
- L *is the set of* locations *or* modes*.*
  - Invariants *are provided by a mapping* Inv *of locations to formulas over variables in* V*.*
  - Flows *are given by* F*, which is a mapping of locations to formulas of the form* $\bigwedge_{x \in \dot{X}} \dot{x} = e_x$ *where* e_x *are expressions over* V*.*
- $E \subseteq L \times G \times L$ *are* discrete transitions*, where* G *denotes the set of all formulas with free variables in* $V \cup X'$*. The variables in* X' *refer to the values of the variables in* V *in the post-state.*
- $Init$ *is a mapping of locations to formulas with free variables* V*, which characterizes the* initial condition *to start in the specific location.*

The intuitive semantics of these automaton is as follows: Like in hybrid programs, discrete actions are assumed to be instantaneous. During continuous evolutions, the variables change according to the dynamics of the current locations.

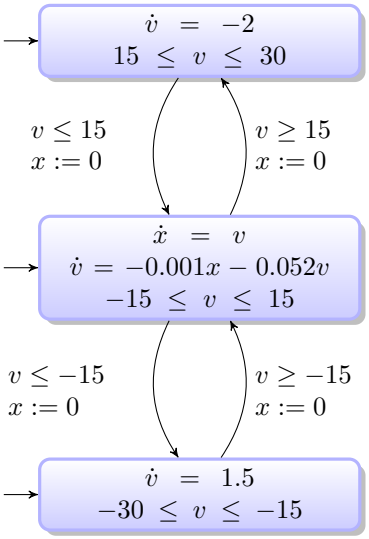


Figure 2.2: Cruise Control

Example 1. *An example for a hybrid automaton is given in Figure 2.2. The automaton models a simple cruise controller. The system has two variables. The variable v is the speed difference to the target speed. Another variable x is used to keep track of the integral error made by the controller. While $v > 15$ the controller chooses maximal deceleration of -2. In case $v < -15$ it chooses maximal acceleration* 1.5*. In between a proportional-integral (PI) controller is used to steer the system smoothly towards a velocity difference of zero. The variable x is reset on each transition in order to avoid preconditioning of the PI controller.*

We give the semantics of hybrid automata in terms of hybrid traces. We assume that a distinct variable ℓ is used to store the current location. Let π denote the projection to specific components of our hybrid automaton. For example $\pi_V(s)$ gives us the valuation of the variables in state s whereas $\pi_L(s)$ returns the location.

Definition 4 (Semantics of Hybrid Automaton)**.** *The* semantics of a hybrid automaton A *is the set $\tau(A)$ of hybrid traces such that, for every hybrid trace $\sigma = (\sigma_0, \sigma_1, \dots) \in \tau(A)$ holds:*

1. $\sigma_0(0) \models \mathit{Init}$
2. *For all $i > 0$ there is an edge from location $l = \pi_L(\sigma_i)$ to location $l' = \pi_L(\sigma_{i+1})$ and $\pi_V(\sigma_i)$ satisfies the invariant of location l and $\pi_V(\sigma_{i+1})$ satisfies the invariant of location l'. Additionally, the guard on this edge is satisfied by using the values of σ_i for V and $\pi_X(\sigma_{i+1})$ for X'.*
3. *During continuous evolutions from some real-valued time point t to some t' the location stays constant and no discrete computations change any values. That is for all $i > 0$, $\sigma_i : [0, r_i] \to \mathbb{R}^n$, if $r_i > 0$ then σ_i is the solution to the initial value problem defined by $\sigma_i(0) = \sigma_{i-1}(\max(\text{dom } \sigma_{i-1}))$ of the flow formula $F(\pi_L(\sigma_i))$ where the variables in U follow some Lebesgue-measurable function $u(\cdot)$.*

2.3 Logics

After presenting different models to describe the operational behavior of hybrid systems, an important issue is how to formally capture properties of those. Mathematical descriptions of system properties have been studied in algebra and mathematical logic. We pursue the latter approach in this work though we borrow ideas and results from algebra.

2.3.1 Propositional Logic

Propositional logic [Fit96] is a classic logic introduced by philosophers to formalize propositions made in natural language. It is thus used to give a mathematical meaning to day-to-day discussions.

Syntax. Let Ξ be a set of atomic formulas.

- true and false are propositional formulas.
- If p is in Ξ, then p is a propositional formula.
- If φ and ψ are propositional formulas, then $\varphi \wedge \psi$ is a propositional formula.
- If φ is a propositional formula then $\neg\varphi$ is a propositional formula.

With these definitions we can add abbreviations for the other classical Boolean combinations.

- $\varphi \vee \psi$ can be defined as $\neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi$ can be defined as $\neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi$ can be defined as $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

Semantics. For an *interpretation* of the atomic formulas I the semantics of propositional formulas is given by the consequence relation $\models$ inductively defined as:

- $I \models true$
- $I \not\models false$
- $I \models p$ iff $I(p) = true$ where $p \in \Xi$
- $I \models \varphi \wedge \psi$ iff $I(\varphi)$ and $I(\psi)$
- $I \models \neg\varphi$ iff $I \not\models \varphi$

2.3.2 First-Order Logic

Even though propositional logic enables us to formalize many sentences, some relations are beyond its expressiveness. A well known example is said to be given by the famous Greek philosopher Aristoteles. He concluded from "All humans are mortal." and that "Sokrates is human." that "Sokrates is mortal.". However, this conclusion cannot be made with pure propositional reasoning, as there is no connection between the first two statements. Mathematically it is necessary to introduce quantifiers and an object domain in order to provide a connection between universal statements ("For all/All...") and constants ("Sokrates"). *First-Order Logic* (FOL) [Fit96] is an extension of propositional logic introducing functions, predicates, and quantifiers, and enables us to capture the above example formally.

The syntax of FOL consists of a countably infinite set of variables Var, the common logic junctors $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$ as well as quantifiers $(\forall, \exists)$. Additionally, there are non-logic symbols supplied by the signature. That is, FOL formulas are built from the following *grammar*

$$\theta ::= x \mid f(\theta_1, \ldots, \theta_n)$$
$$\phi ::= p(\theta_1, \ldots, \theta_n) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x\ \phi \mid \exists x\ \phi$$

where x, x_i are variables, f is a function symbol, θ, θ_i are terms, p is a predicate symbol, and ϕ and ψ are first-order formulas. We now define the *semantics* of first-order logic.

A *structure* $\mathcal{M}$ for the signature is a pair of two sets $(D_\mathcal{M}, \jmath_\mathcal{M})$. The first is non-empty and is called the universe. The latter is the interpretation. An *interpretation* $\jmath$ is a function that maps every function symbol $f \in Func$ to a function with arity n:

$$\jmath_\mathcal{M}(f) : D_\mathcal{M}^n \rightarrow D_\mathcal{M}$$

The same holds for predicate symbols, but the range of those is Boolean.

$$\jmath_\mathcal{M}(p) : D_\mathcal{M}^n \rightarrow \mathbb{B}$$

A *valuation* ν is a function that maps every variable to an element in the universe. That is,

$$\nu : Var \rightarrow D_\mathcal{M}\ .$$

A *semantic modification of a valuation* ν is written as $\nu[x \mapsto d]$ which is a valuation identical to ν except for the valuation of x, which is $d \in D_\mathcal{M}$.

Definition 5 (Semantics of terms)**.** *The* semantics of a term θ *depending on an interpretation* $\jmath$ *and a valuation of variables* ν *is given by a valuation function* $val_{\jmath,\nu}(\theta)$*. This function is inductively defined by*

$$val_{\jmath,\nu}(x) = \nu(x) \text{ iff } x \text{ is a variable}$$
$$val_{\jmath,\nu}(f(\theta_1, \ldots, \theta_n)) = \jmath(f)(val_{\jmath,\nu}(\theta_1), \ldots, val_{\jmath,\nu}(\theta_n))$$

Definition 6 (Semantics of formulas)**.** *The* semantics of a formula ϕ *depending on an interpretation* $\jmath$ *and a valuation of variables* ν *is inductively*

defined as follows:

$$\begin{aligned}
\jmath, \nu \models p(\theta_1, \ldots, \theta_n) \quad &\textit{iff} \quad \jmath(p)(val_{\jmath,\nu}(\theta_1), \ldots, val_{\jmath,\nu}(\theta_n)) = \textit{true} \\
\jmath, \nu \models \neg\phi \quad &\textit{iff} \quad \jmath, \nu \not\models \phi \\
\jmath, \nu \models \phi \wedge \psi \quad &\textit{iff} \quad \jmath, \nu \models \phi \textit{ and } \jmath, \nu \models \psi \\
\jmath, \nu \models \phi \vee \psi \quad &\textit{iff} \quad \jmath, \nu \models \phi \textit{ or } \jmath, \nu \models \psi \\
\jmath, \nu \models \phi \rightarrow \psi \quad &\textit{iff} \quad \jmath, \nu \not\models \phi \textit{ or } \jmath, \nu \models \psi \\
\jmath, \nu \models \phi \leftrightarrow \psi \quad &\textit{iff} \quad \jmath, \nu \models \phi \textit{ if, and only if } \jmath, \nu \models \psi \\
\jmath, \nu \models \forall x\, \phi \quad &\textit{iff} \quad \textit{for all } d \in D_{\mathcal{M}} \textit{ it holds that } \jmath, \nu[x \mapsto d] \models \phi \\
\jmath, \nu \models \exists x\, \phi \quad &\textit{iff} \quad \textit{for one } d \in D_{\mathcal{M}} \textit{ it holds that } \jmath, \nu[x \mapsto d] \models \phi
\end{aligned}$$

We now define substitutions [Fit96].

Definition 7 (Substitution)**.** *A* substitution *is a finite relation between variables and terms, written as* $\Theta = \{x_1 \mapsto \theta_1, \ldots, x_n \mapsto \theta_n\}$*, with* $n \in \mathbb{N}$ *and*

- $x_1, \ldots, x_n$ *are pairwise disjoint variables*
- $\theta_1, \ldots, \theta_n$ *are terms*
- *for all* $i \in \{1, \ldots, n\}$ $x_i \neq \theta_i$ *holds*

Definition 8 (Application of a substitution)**.** *Let* θ *be a term,* ϕ *be a formula, and* $\Theta = \{x_1 \mapsto \theta_1, \ldots, x_n \mapsto \theta_n\}$ *a substitution.*

1. *The* application of Θ to a term θ *results in a new term* $\theta\Theta$*, that results from* θ *by simultaneously replacing all occurrences of the variables* $x_1, \ldots, x_n$ *by the corresponding terms* $\theta_1, \ldots, \theta_n$*.*
2. *The* application of Θ to a formula ϕ *results in a new formula* $\phi\Theta$*, that is created by:*
   a) *renaming all occurrences of a variable* x *in a part of the formula with the form* $\exists x\, \psi$ *or* $\forall x\, \psi$ *where* ψ *is a formula with* x *is one of the* x_i *or occurs in some* θ_i *to a name that does not occur either in* ϕ *nor in the substitution*
   b) *and then replacing all remaining occurrences of the variables* $x_1, \ldots, x_n$ *by the corresponding terms* $\theta_1, \ldots, \theta_n$*.*

2.3.3 First-Order Logic over the Reals

Since the halting problem for a Turing machine can be reduced to the satisfaction problem of first-order logic [Coo71], we cannot hope to decide whether a first-order formula is valid. Even though the reachability problem for hybrid systems is undecidable [ACHH92] as well, we like to have a decidable base logic for our formalisms. In the following, we will base most of our formalisms on a fixed interpretation of a set of predicates and functions symbols. We denote by $FOL_{\mathbb{R}}$ the version of first-order logic where the predicate symbols $>, \geq, =, \leq, <, \neq$ are interpreted in their usual way on terms build from rational constants, variables, and function symbols $+, -, \cdot$ with their usual interpretation. This interpretation here is fixed. So in contrast to classical first-order logic the validity of a formula solely depends on the valuation of the variables.

That is, first-order formulas over the reals are produced by the following grammar:

$$\begin{aligned}\theta ::= & q \mid x \mid (-\theta) \mid (\theta_1 + \theta_2) \mid (\theta_1 - \theta_2) \mid (\theta_1 \cdot \theta_2)\\ p ::= & \theta_1 > \theta_2 \mid \theta_1 \geq \theta_2 \mid \theta_1 = \theta_2 \mid \theta_1 \neq \theta_2 \mid \theta_1 \leq \theta_2 \mid \theta_1 < \theta_2\\ F ::= & p \mid \neg F \mid F \wedge G \mid F \vee G \mid F \rightarrow G \mid F \leftrightarrow G \mid \forall x\ F \mid \exists x\ F\end{aligned}$$

where q is a rational number, x is a variable, θ, θ_i are terms, p is a predicate expression, and F and G are formulas.

The *semantics* is defined like for first-order logic with a fixed domain and fixed interpretations. Therefore, the valuations of terms is just a lifting of the function symbols and rational numbers to the semantics domain. The only thing that is replaced by the valuation function are the variables that are replaced by their real value. "Real" here refers to both the fact that is the actual value that is represented by the variable and the domain the value is drawn from which are the real numbers.

Definition 9 (Semantics of terms)**.** *The* semantics of a term θ *depending on a valuation of variables* $\nu : V \to \mathbb{R}$ *is given by a valuation func-*

tion $val_\nu(\theta)$. This function is inductively defined by

$$\begin{aligned} val_\nu(q) &= q \text{ iff } q \text{ is a rational number} \\ val_\nu(x) &= \nu(x) \text{ iff } x \text{ is a variable} \\ val_\nu(-\theta_1) &= -val_\nu(\theta_1) \\ val_\nu(\theta_1 + \theta_2) &= val_\nu(\theta_1) + val_\nu(\theta_2) \\ val_\nu(\theta_1 - \theta_2) &= val_\nu(\theta_1) - val_\nu(\theta_2) \\ val_\nu(\theta_1 \cdot \theta_2) &= val_\nu(\theta_1) \cdot val_\nu(\theta_2) \ . \end{aligned}$$

Observe that, while the constants appearing in the formulas have to be rationals, the variable values are real numbers. Therefore, we can also express algebraic numbers using this logic. For example $x^2 - 2 = 0 \wedge x > 0$ expresses that $x = \sqrt{2}$. However, the logic provides no means to express numbers like e or π. Because the interpretation of function and predicate symbols is fixed anyway we will from now on use a less verbose notation. That is, by abuse of notation we just write $\nu(\theta)$ for the valuation of a term θ.

Further, we should note that this logic does only consider total functions and therefore there is no division operator here. This is, we consider fractions as abbreviations for multiplications with additional constraints ensuring that the denominators are non-zero.

Example 2. *For instance, the formula*

$$\forall x \ (\frac{a}{x} = b) \tag{2.5}$$

will be interpreted as

$$\forall x \ (x \neq 0 \rightarrow a = xb) \ . \tag{2.6}$$

For existential quantifiers we use conjunctions instead of implications in order to make sure that the denominators are non-zero. That is, the formula

$$\exists x \ (\frac{a}{x} = b) \tag{2.7}$$

will be interpreted as

$$\exists x \ (x \neq 0 \wedge a = xb) \ . \tag{2.8}$$

Note that, for mixed quantifier prefixes this gets quite intricate. Consider the following formula:

$$\forall x\, \exists y\, (\frac{a}{x+y} = \frac{b}{x+1} + \frac{c}{xy}) \tag{2.9}$$

In this case, we have to add constraints that ensure that neither $x+y$, $x+1$, nor xy are 0. The following formula governs all these cases:

$$\begin{aligned}&\forall x\, (x \neq 0 \wedge x+1 \neq 0) \rightarrow \\ &\quad \exists y\, (x+y \neq 0 \wedge xy \neq 0 \wedge a(x+1)xy = b(x+y)xy + c(x+y)(x+1))\end{aligned} \tag{2.10}$$

Observe that, it is necessary to exclude the case $x = 0$ in order to ensure that we can find some y such that $xy \neq 0$ for all x.

Like for the terms, the semantics of formulas is straightforward. The interpretations of the predicate and function symbols are fixed and as expected. The logical connectives and quantifiers are interpreted like in general first-order logic with the sole difference that the domain of the quantifiers is always the real numbers.

Definition 10 (Semantics of formulas)**.** *The* semantics of a formula ϕ *depending on a valuation of variables ν is inductively defined as follows.*

$$\begin{aligned}
\nu \models \theta_1 \sim \theta_2 \quad &\textit{iff} \quad \nu(\theta_1) \sim \nu(\theta_2) \textit{ for } \sim \in \{>, \geq, =, \neq, \leq, <\} \\
\nu \models \neg\phi \quad &\textit{iff} \quad \nu \not\models \phi \\
\nu \models \phi \wedge \psi \quad &\textit{iff} \quad \nu \models \phi \textit{ and } \nu \models \psi \\
\nu \models \phi \vee \psi \quad &\textit{iff} \quad \nu \models \phi \textit{ or } \nu \models \psi \\
\nu \models \phi \rightarrow \psi \quad &\textit{iff} \quad \nu \not\models \phi \textit{ or } \nu \models \psi \\
\nu \models \phi \leftrightarrow \psi \quad &\textit{iff} \quad \nu \models \phi \textit{ if, and only if } \nu \models \psi \\
\nu \models \forall x\, \phi \quad &\textit{iff} \quad \textit{for all values } r \in \mathbb{R} \textit{ it holds that } \nu[x \mapsto r] \models \phi \\
\nu \models \exists x\, \phi \quad &\textit{iff} \quad \textit{for one } r \in \mathbb{R} \textit{ it holds that } \nu[x \mapsto r] \models \phi
\end{aligned}$$

2.3.4 Differential Dynamic Logic d$\mathcal{L}$

In order to link the operational behavior of hybrid systems and their properties, Platzer [Pla10b] proposed a logic called *differential dynamic logic* d$\mathcal{L}$.

The logic is based on the formalism of hybrid programs and strongly inspired by Harel's first-order dynamic logic [HKT00].

Terms and predicate expressions in differential dynamic logic are built the same like in $\text{FOL}_{\mathbb{R}}$. That is, they are defined by the following grammar:

$$\theta ::= q \mid x \mid (-\theta) \mid (\theta_1 + \theta_2) \mid (\theta_1 - \theta_2) \mid (\theta_1 \cdot \theta_2)$$
$$p ::= \theta_1 > \theta_2 \mid \theta_1 \geq \theta_2 \mid \theta_1 = \theta_2 \mid \theta_1 \neq \theta_2 \mid \theta_1 \leq \theta_2 \mid \theta_1 < \theta_2$$

As before, q is a rational number and x is a variable. Now, the $\mathsf{d}\mathcal{L}$-*formulas* are defined by the following *grammar* (ϕ and ψ are formulas, α is a hybrid program):

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

The formulas are designed as an extension of *first-order logic over the reals* with built-in correctness statements about hybrid programs. They can contain propositional connectives $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$ and real-valued quantifiers $\forall, \exists$ for quantifying over parameters and evolution times. For a hybrid program α, $\mathsf{d}\mathcal{L}$ provides correctness statements like $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, where $[\alpha]\phi$ expresses that all traces of system α lead to states in which condition ϕ holds. Likewise, $\langle\alpha\rangle\phi$ expresses that there is at least one trace of α to a state satisfying ϕ. Note that $\mathsf{d}\mathcal{L}$ is closed under logical connectives and quantification. Thus, it provides conditional correctness statements like $\phi \rightarrow [\alpha]\psi$, saying that α satisfies ψ if condition ϕ holds at the initial state, or even nested statements like the reactivity statement $[\alpha]\langle\beta\rangle\phi$, saying that whatever hybrid program α is doing, hybrid program β can react in some way to ensure ϕ. In addition, $\mathsf{d}\mathcal{L}$ can also express mixed quantified statements like $\exists m\, [\alpha]\phi$ saying that there is a choice of parameter m such that system α always satisfies ϕ, which is useful for determining parameter constraints.

For a term θ let $\nu(\theta)$ provide the value of θ in the state ν. The semantics of $\mathsf{d}\mathcal{L}$-formulas is almost identical to that of $\text{FOL}_{\mathbb{R}}$. In addition, the semantics features reachability expressions over hybrid programs.

Definition 11 (Semantics of $\mathsf{d}\mathcal{L}$ formulas)**.** *The* semantics $\models$ *of a* $\mathsf{d}\mathcal{L}$ *formula with respect to state* ν *uses the standard meaning of first-order logic:*

1. $\nu \models \theta_1 \sim \theta_2$ *iff* $\nu(\theta_1) \sim \nu(\theta_2)$ *for* $\sim \in \{<, \leq, =, \geq, >\}$
2. $\nu \models \phi \wedge \psi$ *iff* $\nu \models \phi$ *and* $\nu \models \psi$*, and accordingly for* $\neg, \vee, \rightarrow, \leftrightarrow$

3. $\nu \models \forall x\, \phi$ *iff* $\omega \models \phi$ *for all* ω *that agree with* ν *except for the value of* x

4. $\nu \models \exists x\, \phi$ *iff* $\omega \models \phi$ *for some* ω *that agrees with* ν *except for the value of* x

It extends to correctness statements about a hybrid program α *as follows*

5. $\nu \models [\alpha]\phi$ *iff* $\omega \models \phi$ *for all* ω *with* $(\nu, \omega) \in \rho(\alpha)$

6. $\nu \models \langle\alpha\rangle\phi$ *iff* $\omega \models \phi$ *for some* ω *with* $(\nu, \omega) \in \rho(\alpha)$

Observe that the semantics is defined based on the reachable states of the program and not based on the trace that led to that state. For a variant of the logic that talks about temporal properties of programs in addition to reachability we like to refer the reader to literature about differential temporal dynamic logic (dTL) [Pla07c, Pla10b]. The logic dTL allows for the combination of the box and diamond operators of dynamic logic with those of temporal logic. That it can express statements like "for all runs of system α it is always the case that ϕ holds".

2.4 Metrics, Norms, and Distances

As we aim at comparing hybrid system behaviors, we need notions of distances between values occurring during the execution of these systems. Therefore, we recapitulate mathematical notions of distances, norms, and metrics. To measure distances in an n-dimensional space, different metrics are available.

Definition 12 (Metric [For08])**.** *A* metric *is a function*

$$d : X \times X \to \mathbb{R}$$

that satisfies the following conditions:

1. $d(x, y) = 0$ *iff* $x = y$

2. *Symmetric:* $d(x, y) = d(y, x)$

3. *Triangle inequality:* $d(x, y) \leq d(x, z) + d(z, y)$

Definition 13 (Norm [For08])**.** *For a vector space V, a* norm *is a function*

$$||\cdot|| : V \to \mathbb{R}$$

with the following properties:

1. $||x|| = 0$ *iff* $x = 0$
2. Linear scalability*:* $||\lambda x|| = |\lambda| \cdot ||x||$ *for all* $\lambda \in \mathbb{R}$, $x \in V$
3. Triangle inequality*:* $||x + y|| \leq ||x|| + ||y||$ *for all* $x, y \in V$

Note that given a normed vector space $(V, ||\cdot||)$ we can construct a metric on V as $||x - y||$ for all $x, y \in V$.

In the following, we will discuss some properties of norms. Like the absolute value function, norms are symmetric w.r.t. subtraction in the vector space.

Lemma 1. *Every norm*

$$||\cdot|| : V \to \mathbb{R}$$

satisfies the following property:

$$||x - y|| = ||y - x||$$

Proof. From Property 2 we can derive that

$$\begin{aligned} & ||y - x|| \\ = & || - 1(x - y)|| \\ = & | - 1| \, ||x - y|| \\ = & ||x - y|| \ . \end{aligned}$$

□

Further, we can get an alternative form of the triangle inequality that we will later need for some of our proofs.

Lemma 2. *Every norm*

$$||\cdot|| : V \to \mathbb{R}$$

satisfies the following property:

$$| \, ||x|| - ||y|| \, | \leq ||x - y|| \text{ for all } x, y \in V$$

Proof. We apply Property 3 (the triangle inequality) twice.

$$\begin{aligned} & ||x|| = ||(x-y)+y|| \leq ||x-y|| + ||y|| \\ \Leftrightarrow \quad & ||x|| - ||y|| \leq ||x-y|| \ . \end{aligned} \tag{2.11}$$

Further using Lemma 1,

$$\begin{aligned} & ||y|| = ||x+(y-x)|| \leq ||x|| + ||y-x|| = ||x|| + ||x-y|| \\ \Leftrightarrow \quad & ||y|| - ||x|| \leq ||x-y|| \ . \end{aligned} \tag{2.12}$$

Combining lines (2.11) and (2.12) we get that

$$\max\{||x|| - ||y||, ||y|| - ||x||\} \leq ||x-y|| \ .$$

To conclude the proof, we use that

$$\max\{||x|| - ||y||, ||y|| - ||x||\} = |\,||x|| - ||y||\,| \ .$$

Which gives us the result

$$|\,||x|| - ||y||\,| \leq ||x-y|| \ .$$

□

The two most well-known norms are the Euclidean and the maximum norm. We now give their formal definitions.

Definition 14. *For a vector $x = (x_1, \ldots, x_n) \in V$, we denote by $||\cdot||_e$ the* Euclidean norm*, where*

$$||x||_e = \sqrt{\sum_{i=1}^{n} x_i^2} \ .$$

Definition 15. *For a vector $x = (x_1, \ldots, x_n) \in V$, we denote by $||\cdot||_\infty$ the* maximum norm *by*

$$||x||_\infty = \max\{|x_i| \mid i \in [1,n]\} \ .$$

Later, we will need an additional property that some norms possess. That is the property of monotonicity. We use the Kronecker delta to construct vectors that have exactly one component being 1 and all others being 0. The *Kronecker delta* is defined as

$$\delta_{ij} := \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} .$$

Further, we use $(\delta_{ij})_n$ to denote the n-dimensional vector with component indexed by j, i.e., $(\delta_{ij})_n := (\delta_{i1}, \ldots, \delta_{in})$. For instance, the vector $(0, 0, 1, 0)$ can thus be concisely expressed as $(\delta_{3j})_4$.

Definition 16 (Properties of Norms)**.**

1. *We say a norm is* absolute *iff* $||x|| \leq || \, |x| \, ||$ *f.a.* $x \in V$ *where* $|\cdot|$ *is applied componentwise.*
2. *We say a norm is* monotone in the classical sense *iff* $|x| \leq |y|$ *implies that* $||x|| \leq ||y||$ *f.a.* $x, y \in V$ *where* $\leq$ *is interpreted componentwise on vectors.*
3. *We say a norm is* monotone

$$\text{if } ||(x_1, \ldots, x_n)|| \leq \delta \text{ then } |x_i| \leq \frac{\delta}{||(\delta_{ij})_n||} \text{ f.a. } i \in \{1, \ldots, n\} \ . \tag{2.13}$$

Johnson and Nylen [JN91] showed that all these properties are equivalent.

Example 3. *As one can easily see, the property* (2.13) *is satisfied for the Euclidean and the maximum norm.*

Let $(x_1, \ldots, x_n)$ *be a vector such that* $||(x_1, \ldots, x_n)||_\infty \leq \delta$*. Further, assume that for some* x_i *it holds that* $|x_i| > \frac{\delta}{||(\delta_{ij})_n||_\infty}$*. Observe that,* $||(\delta_{ij})_n||_\infty = 1$*. Thus, we can simplify this expression to* $|x_i| > \delta$*. This, however, is a contradiction to the assumption that*

$$\max\{|x_i| \mid i \in [1, n]\} \leq \delta \ .$$

Therefore, the maximum norm satisfies property (2.13)*.*

Consider the Euclidean norm. First, observe that again $||(\delta_{ij})_n||_e = 1$. *We assume that* $(x_1, \ldots, x_n)$ *is a vector with* $||(x_1, \ldots, x_n)||_e \leq \delta$. *Further, we assume that for some* x_i *it holds that* $|x_i| > \delta$. *Thus, we get*

$$||(x_1, \ldots, x_n)||_e = \sqrt{\sum_{j=1}^{i-1} x_j^2 + x_i^2 + \sum_{j=i+1}^{n} x_j^2} \leq \delta \ .$$

As the square root is monotonic, i.e. for $x \geq y \geq 0$ *it holds that* $\sqrt{x} \geq \sqrt{y}$, *and for all* x_j^2 *we know that* $x_j^2 \geq 0$ *we consider the case where all* $x_j = 0$. *Now we can simplify the expression to* $\sqrt{x_i^2} \leq \delta$. *However this is equivalent to* $|x_i| \leq \delta$ *which is a contradiction to our assumption. Thus, the Euclidean norm satisfies property* (2.13).

A non-monotone norm can be easily constructed as well.

Example 4. *We construct a non-monotone norm on* $\mathbb{R}^2$. *Let* $c > 0$.

$$||(x, y)|| = \max\{c|x + y|, |x - y|\}$$

This norm is not monotone as $||(1, 1)|| = 2c$ *whereas* $||(1, -2)|| = 3$ *although obviously* $|1| < |-2|$. *We now show that this function is indeed a norm.*

1. $||(0, 0)|| = \max\{c|(0 + 0)|, |(0 - 0)|\} = 0$. *Assume* $||(x, y)|| = 0$. *This means* $0 = \max\{c|(x + y)|, |(x - y)|\}$. *However, the sum and the difference of two numbers can only be* 0 *if both numbers are* 0.

2. *Linear scalability:*

$$\begin{aligned} ||(\lambda x, \lambda y)|| &= \max\{c|\lambda x + \lambda y|, |\lambda x - \lambda y|\} \\ &= \max\{c|\lambda(x + y)|, |\lambda(x - y)|\} \\ &= \max\{c|\lambda| \cdot |x + y|, |\lambda| \cdot |x - y|\} \\ &= |\lambda| \max\{c|x + y|, |x - y|\} \end{aligned}$$

3. *Triangle inequality:*

$$\begin{aligned} &||(x_1 + x_2, y_1 + y_2)|| = \\ &\qquad \max\{c|(x_1 + x_2) + (y_1 + y_2)|, |(x_1 + x_2) - (y_1 + y_2)|\} \end{aligned}$$

- *Assume* $c|(x_1 + x_2) + (y_1 + y_2)| \geq |(x_1 + x_2) - (y_1 + y_2)|$*, then*

$$\begin{aligned} c|(x_1 + x_2) + (y_1 + y_2)| &= c|(x_1 + y_1) + (x_2 + y_2)| \\ &\leq c|(x_1 + y_1)| + c|(x_2 + y_2)| \\ &\leq \quad \max\{c|x_1 + y_1|, |x_1 - y_1|\} \\ &\quad + \max\{c|x_2 + y_2|, |x_2 - y_2|\} \\ &= ||(x_1, y_1)|| + ||(x_2, y_2)|| \ . \end{aligned}$$

- *Assume* $c|(x_1 + x_2) + (y_1 + y_2)| < |(x_1 + x_2) - (y_1 + y_2)|$*, then*

$$\begin{aligned} |(x_1 + x_2) - (y_1 + y_2)| &= |(x_1 - y_1) + (x_2 - y_2)| \\ &\leq |x_1 - y_1| + |x_2 - y_2| \\ &\leq \quad \max\{c|x_1 + y_1|, |x_1 - y_1|\} \\ &\quad + \max\{c|x_2 + y_2|, |x_2 - y_2|\} \\ &= ||(x_1, y_1)|| + ||(x_2, y_2)|| \ . \end{aligned}$$

In the following lemma, we formulate a well known result from functional analysis. That is, on a finite dimensional vector space all norms are equivalent.

Lemma 3. *On a finite dimensional vector space any two norms* $||\cdot||_a$ *and* $||\cdot||_b$ *are* equivalent*, i.e., there is* $\lambda > 1$ *such that for all* $x \in V$

$$\frac{1}{\lambda}||x||_a \leq ||x||_b \leq \lambda||x||_a \ .$$

Proof. See for example [Con90]. □

Using this result, we can later give transformations such that we can use the monotonicity property given in Definition 16 even though the original norm might not possess this property.

CHAPTER THREE

Similarity

Any man, in the right situation, is capable of murder. But not any man is capable of being a good camper. So, murder and camping are not as similar as you might think.

— Jack Handy

Contents

"Those are similar." is easily stated but seldom meant in a formal way. In this chapter we try to formalize what similarity means for complex physical systems modeled mathematically as hybrid systems. We define families of

quantitative refinement notions. These notions can be considered as *robust refinement relations* as they allow for bounded perturbations w.r.t. to variable values (in the following called spatial deviations) or timings (in the following referred to as temporal deviations). However, in contrast to classical refinement this means that not all metric temporal logic properties are preserved. Therefore, we study which properties can actually be transferred from a system to its robust refinements.

Contributions. We define two families of robust refinement relations that provide quantitative notions of robust refinement of hybrid systems. We prove that stability properties are preserved in a relaxed sense by our notion of robust refinement. Subsequently, we define a logic suitable to express properties of hybrid systems and present a syntactic transformation to compute which formulas are satisfied by all robust refinements of a given system.

Structure of the Chapter. In Section 3.1 we provide two families of robust refinement relations. Section 3.2 covers our study of what properties are preserved under refinement. Related work is discussed in Section 3.3. We conclude the chapter in Section 3.4.

3.1 Notions of Robust Refinement

In day to day life, we consider two things similar if they look almost the same, or behave almost the same. In our mathematical model, we are interested in similar behavior as that is what we use to measure whether our system is acting correctly in some sense.

The behavior of hybrid systems is defined in terms of variable valuations changing over time. In the previous chapter, we defined the semantics of hybrid systems with regard to a two-dimensional time model. However, in the physical world at a single instant in time at most one valuation of the variables will be observable. We consider the other steps internal.

Therefore, we have to define what it means for piecewise continuous trajectories to have similar behavior. We can get a first measure on how similar the systems are by computing the distance of these variable valuations at each point in time. Furthermore, we could compute the maximum over these measurements and get a value characterizing the similarity of

the two systems. However, for real-time systems and, thus, for hybrid systems the instantaneous changes to the variable valuations on transitions can make this distance rather huge.

Example 5. *The following program starts with some arbitrary value for x. The value of x then evolves with constant slope of one until it reaches 100 upon which it is reset to 0.*

$$\dot{x} = 1 \,\&\, x \leq 100; ?x = 100; x := 0; \dot{x} = 0$$

Now let us see what happens if we start this system with initial value $x = 0$. That means it takes exactly 100 time units to reach the guard $?x = 100$. Started at value $x = 1$, we get a different trajectory. The system will now only need 99 time units to reach this guard and the distance to the other trajectory will be 1 up until that point. However, at the moment the system jumps to 0 the distance suddenly becomes 99 and will grow to 100 within one time unit. At that point, the distance will, due to the reset in the trajectory started at $x = 0$, become zero again.

Intuitively, we only did a slight variation in the initial value and would expect the results to be similar. Looking at the trajectory plots (cf. Figure 3.1) the two also look similar but our informally defined notion measures a huge distance between them.

To account for this effect, we allow for some bounded deviation regarding the time of the points we are comparing. This means given a valuation of the variables of one system at some point in time, there is within close distance in time a point where the valuation of the variables of the other system is close. We restrict the temporal distance by a function ε and the spatial distance by another function δ. If these functions are constant, we denote, by abuse of notion, their value just by ε and δ. For the temporal deviations, we use ε-retimings to relate the time axes of the program semantics. These retimings allow stretching and compressing of the time axes we want to compare.

First, we recapitulate two notions. We say a relation $R : X \times X$ is *left-total* iff for all $x \in X$ there is $y \in X$ such that $(x, y) \in R$. Further, it is *surjective* iff for all $y \in X$ there is $x \in X$ such that $(x, y) \in R$. Relations with both of these properties are shown for a finite domain in Figure 3.2. The relations are left-total as an edge starts at every node on the left. They are surjective as the same holds for the nodes on the right. If we, further, assume the existence of an order on X then we can see that not

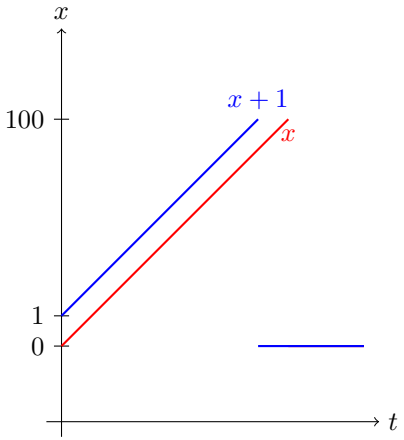


Figure 3.1: Plot of two simple systems with almost identical behavior (cf. Example 5). Note that the axes do not scale.

every *bijection* (cf. Figure 3.2a) is order-preserving. However, there exists a relation between the two sets that is (cf. Figure 3.2b). Obviously, the canonical bijection would be order-preserving too.

For our retimings we want to exclude the situation depicted in Figure 3.2a as we want to preserve the order of events.

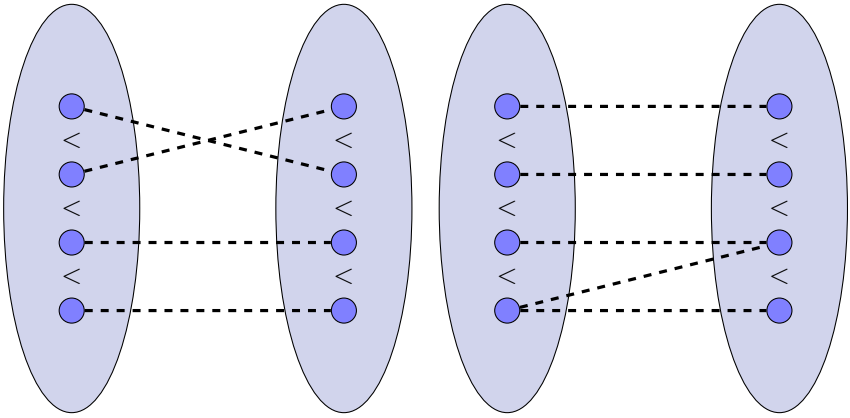
Definition 17 (Retiming)**.** *A left-total, surjective relation* $\mathfrak{r} \subseteq \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ *is called* retiming *iff*

$$\forall (t, \tilde{t}) \in \mathfrak{r}\, \forall (t', \tilde{t}') \in \mathfrak{r} : ((t < t' \to \tilde{t} \leq \tilde{t}') \wedge (\tilde{t} < \tilde{t}' \to t \leq t')) \ .$$

That is, a retiming is a relation between two time axes such that if we have increasing time stamps on the one axes then the time stamps we relate them to are non-decreasing. The definition enforces this property in both directions. Still it allows for relating a single point on either of the axes to a whole interval on the other.

Lemma 4. *If* $\mathfrak{r}$ *is a retiming then* $(0, 0) \in \mathfrak{r}$.

Proof. As $\mathfrak{r}$ is left-total we know that for some $y \in \mathbb{R}_{\geq 0}$ we have that $(0, y) \in \mathfrak{r}$. As it is surjective, we further know that $(x, 0) \in \mathfrak{r}$ for some



(a) Bijection (b) Order preserving relation

Figure 3.2: A left-total, surjective relation on a finite domain

$x \in \mathbb{R}_{\geq 0}$. If we assume $x > 0$, then we can conclude from the retiming property that $y \leq 0$. This however means that $y = 0$ and thus $(0,0) \in \mathfrak{r}$. Assume on the contrary that $x \leq 0$. Hence as $x \in \mathbb{R}_{\geq 0}$ it holds that $x = 0$, and therefore $(0,0) \in \mathfrak{r}$. □

We now add a bound on the distance of time points to compare by defining ε-retimings on top of retimings.

Definition 18 (ε-Retiming)**.** *Let ε be a non-negative real number. A retiming $\mathfrak{r} \subseteq \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ is called* ε-retiming *iff*

$$\forall (t, \tilde{t}) \in \mathfrak{r} : |t - \tilde{t}| \leq \varepsilon \ .$$

Note that we use a non-strict inequality here in order to have the identity relation defined as 0-retiming. An example for a retiming/ε-retiming is depicted in Figure 3.3. In order to receive the retiming from the picture we assume that all points connected by the dashed lines are in relation and there are monotone functions connecting the intervals inbetween. This is we could use the canonical bijection to relate for example the interval $[0, 1]$ on the t-axis and $[0, \frac{1}{2}]$ on the $\tilde{t}$-axis.

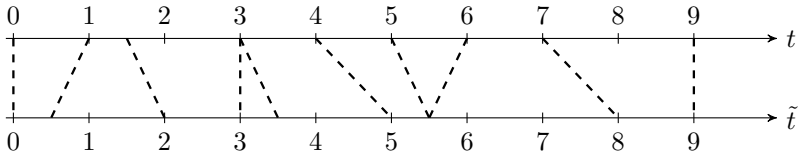


Figure 3.3: Example for a retiming relation $\mathfrak{r}$

Definition 19. *Let $\varepsilon_1, \varepsilon_2$ be non-negative real numbers. Let $\mathfrak{r}_1$ be an ε_1-retiming and $\mathfrak{r}_2$ be an ε_2-retiming. We define the classical relation composition on retimings as*

$$\mathfrak{r}_1 \oplus \mathfrak{r}_2 := \{(t, t') \mid \exists s \bullet (t, s) \in \mathfrak{r}_1 \wedge (s, t') \in \mathfrak{r}_2\} \ . \tag{3.1}$$

Further, we define the set of possible combinations *as*

$$\mathfrak{r}_1 \circ \mathfrak{r}_2 := \{\mathfrak{r} \mid \mathfrak{r} \textit{ is an } (\varepsilon_1 + \varepsilon_2)\textit{-retiming} \wedge \mathfrak{r} \subseteq \mathfrak{r}_1 \oplus \mathfrak{r}_2\} \ . \tag{3.2}$$

Observe that the classical relation composition, denoted by $\oplus$ here, does not yield a retiming. This is, because the resulting relation might not respect the order. Therefore, we define an additional composition operator $\circ$ that contains all subsets that are indeed retimings. Furthermore, note that unlike function composition our notion of retiming composition denotes a set of possible compositions as the composition is not uniquely determined.

In order to be sure that this composition of retimings is well-defined we have to show that the composition is non-empty. For this, we first show that given an ε_1-retiming $\mathfrak{r}_1$, and an ε_2-retiming $\mathfrak{r}_2$, the distance of related points in $\mathfrak{r}_1 \oplus \mathfrak{r}_2$ is always bounded by $\varepsilon_1 + \varepsilon_2$.

Lemma 5. *Let $\mathfrak{r}_1$ be an ε_1-retiming, and $\mathfrak{r}_2$ be an ε_2-retiming. Then for all pairs $(t, t') \in \mathfrak{r}_1 \oplus \mathfrak{r}_2$ it holds that $|t - t'| \leq \varepsilon_1 + \varepsilon_2$.*

Proof. By definition, for all $(t, t') \in \mathfrak{r}_1 \oplus \mathfrak{r}_2$ it holds that there is some s such that $(t, s) \in \mathfrak{r}_1$ and $(s, t') \in \mathfrak{r}_2$. As $\mathfrak{r}_1$ is an ε_1-retiming we get that $|t - s| \leq \varepsilon_1$. Further, as $\mathfrak{r}_2$ is an ε_2-retiming it holds that $|s - t'| \leq \varepsilon_2$. We now combine these two inequalities which gives

$$|t - s| + |s - t'| \leq \varepsilon_1 + \varepsilon_2 \ .$$

Using the triangle inequality we get that

$$|t - t'| = |(t - s) + (s - t')| \leq |t - s| + |s - t'| \leq \varepsilon_1 + \varepsilon_2 \ .$$

This concludes the proof. □

Lemma 6. *Let $\mathfrak{r}_1$ be an ε_1-retiming and $\mathfrak{r}_2$ be an ε_2-retiming. The set of possible compositions of $\mathfrak{r}_1$ and $\mathfrak{r}_2$ is non-empty, i.e., $\mathfrak{r}_1 \circ \mathfrak{r}_2 \neq \emptyset$.*

Proof. Let $\mathfrak{r}_1$ be an ε_1-retiming and $\mathfrak{r}_2$ be an ε_2-retiming. We now have to show that $\mathfrak{r}_1 \oplus \mathfrak{r}_2$ contains an $(\varepsilon_1 + \varepsilon_2)$-retiming.

For a point t let $r^{-1}(t)$ denote the pre-image of the relation $\mathfrak{r}_1$, i.e.,

$$r^{-1}(t) = \{t' \mid (t', t) \in \mathfrak{r}_1\} \ .$$

Further, let $r(t)$ be the image of $\mathfrak{r}_2$, i.e.,

$$r(t) = \{t' \mid (t, t') \in \mathfrak{r}_2\} \ .$$

We now construct a subset $R := \bigcup_{t \in \mathbb{R}_{\geq 0}} R(t)$ of $\mathfrak{r}_1 \oplus \mathfrak{r}_2$. For a given $t \in \mathbb{R}_{\geq 0}$, we construct $R(t)$ according to the following algorithm.

- If $r^{-1}(t)$ is a singleton set then $(r^{-1}(t), a) \in R(t)$ for all $a \in r(t)$.
- Otherwise, $r^{-1}(t)$ is countably infinite.
  - If $r(t) = \{a\}$ is a singleton set then we map all points in $r^{-1}(t)$ to this point. That is $(a, b) \in R(t)$ for $b \in r^{-1}(t)$.
  - Otherwise, we construct an order preserving relation the following way:
    1. If the topology of $r^{-1}(t)$ is the same as the topology of $r(t)$ we add the canonical order preserving bijection between the two sets.
    2. If one interval $[a, b[$ is left closed and the other interval $]c, d[$ is left open, then $(a, y) \in R(t)$ for all $y \in]c, \frac{d-c}{2}[$ and all pairs $(x, y) \in R(t)$ from the canonical order preserving bijection between $]a, b[$ and $]\frac{d-c}{2}, d[$.
    3. The other cases follow by symmetry. Using the same construction as in case 2.

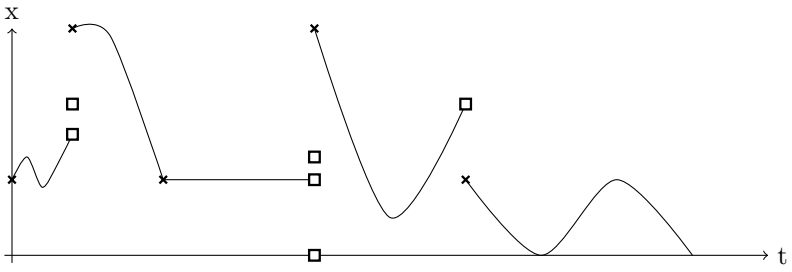


Figure 3.4: Example trajectory

Applying this construction for all $t' \in \mathbb{R}_{\geq 0}$, we can conclude from the fact that $\mathfrak{r}_1$ is surjective and $\mathfrak{r}_2$ is left-total and surjective that the resulting relation R is left-total and surjective as well. Further, it is indeed a retiming as all construction steps are order preserving, i.e., ensure the retiming property.

In order to show that R is indeed an $(\varepsilon_1 + \varepsilon_2)$-retiming, we need to show that for each pair $(a, b) \in R$ the distance between a and b is bounded by $(\varepsilon_1 + \varepsilon_2)$. By construction, $R \subseteq \mathfrak{r}_1 \oplus \mathfrak{r}_2$. From Lemma 5 we know that for all pairs in $\mathfrak{r}_1 \oplus \mathfrak{r}_2$ the distance is bounded by $\varepsilon_1 + \varepsilon_2$. Therefore, this property also holds for all of its subsets, and, thus, for the relation R. Therefore, we can conclude that $R \in \mathfrak{r}_1 \circ \mathfrak{r}_2$ and, finally, that thus the set of possible compositions $\mathfrak{r}_1 \circ \mathfrak{r}_2$ is non-empty which concludes the proof. □

As we allow for multiple discrete actions at a single point in time in our models, systems could differ just because they execute these actions in a different order. However, these transient intermediate variable values occur during calculations that we consider instantaneous and, thus, internal. We therefore do not consider them as observable. That is, we consider variable values to be *observable*, iff they are the starting or intermediate value of a continuous evolution. In Figure 3.4 such a trace is depicted. Hence for this trace the values marked with boxes are omitted. The values marked with crosses remain and denote the valuation of the trace at that instant of time. Of course we keep all valuations unchanged where these are uniquely determined anyway.

Definition 20 (Valuation of Traces)**.** *For a finite trace $\sigma = (\sigma_0, \ldots, \sigma_n)$ or an infinite trace $\sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots)$ we define the* value *of σ at a point t denoted $\sigma(t)$, as follows*

$$\sigma(t) \hat{=} \begin{cases} \sigma_\ell(\mathit{shift}(t, \ell)) & \text{if } \ell \text{ is defined} \\ \mathit{undefined} & \text{otherwise} \end{cases}$$

where $\mathit{shift}(t, i) = t - \sum_{j=0}^{i-1} \max(\mathrm{dom}\ \sigma_j)$ and $\ell = \max\{i \mid \mathit{shift}(t, i) \geq 0\}$.

Observe that, in the previous definition, ℓ is the index of the element of the trace such that either it is the last element or $\sigma_{\ell+1}$ to denote a valuation at time t. Furthermore, we like to point out that these valuations can be seen as the output function of our hybrid systems over a real-valued time axis.

Based on these *observable traces*, we recall the similarity notion we previously defined in [QFD11]. This notion allows for bounded deviations in timing behavior as well as valuations of the system variables.

Definition 21 (Similarity of Traces [QFD11])**.** *For two traces σ_1 and σ_2, given two non-negative real numbers ε, δ, we say that σ_1 is ε-δ-*similar *to σ_2 (denoted by $\sigma_1 \leftarrow\!\!\!\!\!\!\sim^{\varepsilon}_{\delta}\!\!\!\!\!\!\rightarrow \sigma_2$) iff there is an ε-retiming $\mathfrak{r}$ such that*

$$\forall (t, \tilde{t}) \in \mathfrak{r} : ||\sigma_1(t) - \sigma_2(\tilde{t})|| \leq \delta \ .$$

We call $\mathfrak{r}$ the retiming witnessing $\sigma_1 \leftarrow\!\!\!\!\!\!\sim^{\varepsilon}_{\delta}\!\!\!\!\!\!\rightarrow \sigma_2$.

From the fact that the constants provide upper bounds on the temporal and spatial distance we can deduce that for larger values of these constants the traces will also be in that relation.

Lemma 7 (Monotonicity)**.** *Let σ_1, σ_2 be hybrid traces. If $\sigma_1 \leftarrow\!\!\!\!\!\!\sim^{\varepsilon_1}_{\delta_1}\!\!\!\!\!\!\rightarrow \sigma_2$ then $\sigma_1 \leftarrow\!\!\!\!\!\!\sim^{\varepsilon_2}_{\delta_2}\!\!\!\!\!\!\rightarrow \sigma_2$ for all $\varepsilon_2 \geq \varepsilon_1$ and $\delta_2 \geq \delta_1$.*

Remark 1. *For fixed ε, δ, the relation $\leftarrow\!\!\!\!\!\!\sim^{\varepsilon}_{\delta}\!\!\!\!\!\!\rightarrow$ for hybrid traces is reflexive and symmetric, but not transitive.*

We illustrate this remark in Example 6. While the relation does not possess the classical transitivity property, we can still obtain a weak form of transitivity. For this, like in Lemma 5, we have to add up the spatial and temporal bounds respectively. Still, this might give rise to an iterative approach in proving similarity.

Lemma 8 (Weak Transitivity)**.** *If σ_1, σ_2, and σ_3 are hybrid traces such that $\sigma_1 \overset{\varepsilon_1}{\leftarrow\!\!\wr\!\!\rightarrow}{}_{\delta_1} \sigma_2$ and $\sigma_2 \overset{\varepsilon_2}{\leftarrow\!\!\wr\!\!\rightarrow}{}_{\delta_2} \sigma_3$ then $\sigma_1 \overset{\varepsilon_1+\varepsilon_2}{\leftarrow\!\!\wr\!\!\rightarrow}{}_{\delta_1+\delta_2} \sigma_3$.*

Proof. Let $\mathfrak{r}_1$ (respectively $\mathfrak{r}_2$) be a retiming that witnesses $\sigma_1 \overset{\varepsilon_1}{\leftarrow\!\!\wr\!\!\rightarrow}{}_{\delta_1} \sigma_2$ (respectively $\sigma_2 \overset{\varepsilon_2}{\leftarrow\!\!\wr\!\!\rightarrow}{}_{\delta_2} \sigma_3$). Let $\mathfrak{r} \in \mathfrak{r}_1 \circ \mathfrak{r}_2$. From Lemma 6 we know that such an $\mathfrak{r}$ exists.

Let $(t, t') \in \mathfrak{r}$ be arbitrary. From Definition 19 we know that there is s such that $(t, s) \in \mathfrak{r}_1$ and $(s, t') \in \mathfrak{r}_2$.

This gives that

$$||\sigma_1(t) - \sigma_2(s)|| \leq \delta_1$$

and

$$||\sigma_2(s) - \sigma_3(t')|| \leq \delta_2 \ .$$

Adding these inequalities, we get

$$||\sigma_1(t) - \sigma_2(s)|| + ||\sigma_2(s) - \sigma_3(t')|| \leq \delta_1 + \delta_2 \ . \tag{3.3}$$

Using the triangle inequality we get that

$$||\sigma_1(t) - \sigma_3(t')|| = ||\sigma_1(t) - \sigma_2(s) + \sigma_2(s) - \sigma_3(t')|| \tag{3.4}$$

$$||\sigma_1(t) - \sigma_2(s) + \sigma_2(s) - \sigma_3(t')|| \leq ||\sigma_1(t) - \sigma_2(s)|| + ||\sigma_2(s) - \sigma_3(t')|| \ . \tag{3.5}$$

Therefore, by combining (3.3), (3.4), and (3.5) we get

$$||\sigma_1(t) - \sigma_3(t')|| \leq \delta_1 + \delta_2 \ .$$

Using Lemma 5, $\mathfrak{r}$ is an $(\varepsilon_1 + \varepsilon_2)$-retiming. Thus, we conclude that σ_1 and σ_3 are $(\varepsilon_1 + \varepsilon_2)$-$(\delta_1 + \delta_2)$-similar, i.e., $\sigma_1 \overset{\varepsilon_1+\varepsilon_2}{\leftarrow\!\!\wr\!\!\rightarrow}{}_{\delta_1+\delta_2} \sigma_3$. □

This notion for similarity of hybrid traces can now be lifted to a notion of robust refinement for hybrid systems in a straightforward way.

Definition 22 (Refinement of Hybrid Systems [QFD11])**.** *A hybrid system α is an ε-δ-*refinement *of another hybrid system β (denoted by $\alpha \overset{\varepsilon,\delta}{\wr\!\!\rightarrow} \beta$) iff for all traces of $\sigma_\alpha \in \tau(\alpha)$, there is a trace $\sigma_\beta \in \tau(\beta)$ such that $\sigma_\alpha \overset{\varepsilon}{\leftarrow\!\!\wr\!\!\rightarrow}{}_{\delta} \sigma_\beta$ holds.*

Note that we the following idea underlies our choice of symbols for robust refinement and similarity: We use an arrow that is similar to the mapping arrow in order to indicate that there is a mapping between the traces of α and those of β. For hybrid traces, we use a symmetric version of this arrow because the relation is symmetric on traces.

In [QFD11] we used the term *simulation* for this notion of robust refinement. However, it is a notion of trace inclusion and does not fit with the notion used in Milners work [Mil99] on bisimulation notions for process algebras. Therefore, we decided to called it *refinement* here, rather than simulation.

We now turn to studying properties of our notion of robust refinement for hybrid systems. The results from Lemma 7 carry over from traces to systems and we can, thus, deduce the following proposition.

Proposition 1. *Let α, β by hybrid systems. If $\alpha \lesssim_{\varepsilon_1,\delta_1} \beta$ then $\alpha \lesssim_{\varepsilon_2,\delta_2} \beta$ for all $\varepsilon_2 \geq \varepsilon_1$ and $\delta_2 \geq \delta_1$.*

Remark 2. *The relation $\lesssim_{\varepsilon,\delta}$ for hybrid systems is reflexive but neither symmetric nor antisymmetric nor transitive.*

We illustrate the Remarks 1 and 2 in a common example.

Example 6. *Let $x(t) = 1$, $y(t) = 2$, $z(t) = 3$ for $t \in \mathbb{R}_{\geq 0}$ be constant functions. The traces are plotted in Figure 3.5. It is easy to see that $x \lessapprox_{\delta_1} y$ and $y \lessapprox_{\delta_1} z$ for $\delta_1 \hat{=} ||1||$. Further, $y \lessapprox_{\varepsilon} x$ holds for any $\varepsilon \geq 0$. However, the distance between x and z is*

$$\delta_2 \hat{=} ||x - z|| = ||2|| > \delta_1$$

and, thus, it does not hold that $x \lessapprox_{\delta_1} z$. Therefore, the relation is not transitive. Instead $x \lessapprox_{\delta_2} z$ holds.

If we move from traces to systems, we can see that the missing transitivity just carries over.

Let $\alpha_x, \alpha_y, \alpha_z, \alpha_{xy}$ be hybrid systems that produce the sets $\tau(\alpha_x) = \{x(t)\}$, $\tau(\alpha_y) = \{y(t)\}$, $\tau(\alpha_z) = \{z(t)\}$, and $\tau(\alpha_{xy}) = \{x(t), y(t)\}$. As $x \lesssim_{0,\delta_1} y$ we get that $\alpha_x \lesssim_{0,\delta_1} \alpha_y$. Because the relation is symmetric for traces it also holds that $\alpha_y \lesssim_{0,\delta_1} \alpha_x$. However, even though $\alpha_z \lesssim_{0,\delta_1} \alpha_{xy}$ it does not hold that $\alpha_{xy} \lesssim_{0,\delta_1} \alpha_z$. This follows from the fact that $x \lesssim_{0,\delta_1} z$ does not hold.

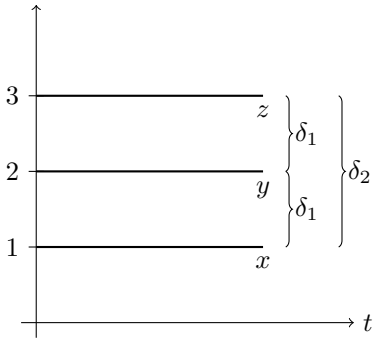


Figure 3.5: Plot of the constant trajectories x, y, z of Example 6.

Like Lemma 7 we can also lift the results from Lemma 8 from traces to programs and arrive at the following proposition. That is the additive transitivity carries over from traces to hybrid systems.

Proposition 2. *Let α, β be hybrid systems. If $\alpha \wr^{\varepsilon_1,\delta_1}\rangle \beta$ and $\beta \wr^{\varepsilon_2,\delta_2}\rangle \gamma$ then $\alpha \wr^{\varepsilon_1+\varepsilon_2,\delta_1+\delta_2}\rangle \gamma$.*

In some cases it is sufficient to use a slightly weaker notion of similarity. This can be obtained by dropping the bound on the temporal distance. If we do not impose an upper bound on the temporal distance, we can still get useful insights. Consider the case where our goal is prove that a system works within certain spatial bounds. In that case the timing behavior is of no interest at all and it is sufficient to show that some retiming exists such that the systems are similar as opposed to an ε-retiming. We now define a notion of similarity based on this observation.

Definition 23. *For two traces σ_1 and σ_2, given a non-negative real number δ, we say that σ_1 is* weakly δ-similar *to σ_2 (denoted by $\sigma_1 \langle\!\!\sim^{\infty}_{\delta}\!\!\rangle \sigma_2$) iff there is a retiming $\mathfrak{r}$ such that*

$$\forall (t, \tilde{t}) \in \mathfrak{r} : ||\sigma_1(t) - \sigma_2(\tilde{t})|| \leq \delta \ .$$

Again, we call $\mathfrak{r}$ the retiming witnessing $\sigma_1 \langle\!\!\sim^{\infty}_{\delta}\!\!\rangle \sigma_2$.

Like before, we lift the definition of similarity from traces to a refinement relation on hybrid systems.

Definition 24. *A hybrid system α is a* weak δ-refinement *of another hybrid system β (denoted by $\alpha \leadsto^{\infty,\delta} \beta$) iff for all traces of $\sigma_\alpha \in \tau(\alpha)$, there is a trace $\sigma_\beta \in \tau(\beta)$ such that $\sigma_\alpha \leftarrow\!\!\!\!\!\!\rightsquigarrow^{\infty,\delta} \sigma_\beta$ holds.*

Note that the notion of weak δ-refinement is strictly weaker than the notion of ε-δ-refinement. As direct consequences of the definitions of ε-δ-similarity (respectively ε-δ-refinement) and weak δ-similarity (respectively weak δ-refinement) we get the following lemmas.

Lemma 9. *Let σ_1 and σ_2 be hybrid traces. If for some ε, δ it holds that $\sigma_1 \leftarrow\!\!\!\!\!\!\rightsquigarrow^{\varepsilon,\delta} \sigma_2$ then it also holds that $\sigma_1 \leftarrow\!\!\!\!\!\!\rightsquigarrow^{\infty,\delta} \sigma_2$.*

Lemma 10. *Let α, β be hybrid systems. If for some ε, δ it holds that $\alpha \leadsto^{\varepsilon,\delta} \beta$ then it also holds that $\alpha \leadsto^{\infty,\delta} \beta$.*

In the following we use the term *similar* sometimes on systems instead of traces to denote the fact that for two systems α and β either $\alpha \leadsto^{\varepsilon,\delta} \beta$ or $\beta \leadsto^{\varepsilon,\delta} \alpha$ for some ε and δ or sometimes just refer to weak δ refinement in one of these directions. Further, we refer to both notions of refinement as *robust refinement*.

3.2 Property Preservation

In this section, we discuss what properties can be transferred between systems that are in refinement relation with respect to the notions defined in Section 3.1. First, we consider different notions of stability and in which form they are preserved. Then we move to a more general setting by defining a logic to formulate properties of systems and present a syntactic transformation to compute which properties are satisfied by all its robust refinements.

3.2.1 Stability and Region Stability

In control engineering, an important property of analog and hybrid systems is stability. There are different notions of stability. The most widely used is that of *Lyapunov stability*. As this notion only makes sense in the setting of infinite traces, we restrict ourselves to those in this section.

A system is considered stable w.r.t. to a point x_0 if for all of its traces it holds that if we fix a ball of diameter $e > 0$ around x_0 then we can find a

possibly smaller ball with diameter $d > 0$ such that every trajectory that starts within this second ball will never leave the first one.

Definition 25 (Stable [Kha96]). *A hybrid trace σ is called* stable w.r.t. a point x_0 *if for any $e > 0$ there is $d > 0$ such that for all $t > 0$ if $||\sigma(0) - x_0|| < d$ then $||\sigma(t) - x_0|| < e$.*

A hybrid system α is called stable w.r.t. a point x_0 *if all its traces $\sigma \in \tau(\alpha)$ are stable w.r.t. x_0.*

Usually stability is combined with a notion of attraction. We say that a point x_0 is attractive for some system if all the trajectories converge towards that point when time goes to infinity.

Definition 26 (Attractive [Kha96]). *A hybrid trace σ is* attracted to the point x_0 *if*

$$\lim_{t \to \infty} ||\sigma(t) - x_0|| = 0 \ .$$

A hybrid system α is attracted to the point x_0 *if all traces $\sigma \in \tau(\alpha)$ are attracted to x_0.*

Combining these two properties, we get the notion of asymptotic stability.

Definition 27 (Asymptotically Stable [Kha96]). *If a hybrid trace is both stable with respect to a point x_0 and attracted to it, it is called* asymptotically stable *with respect to x_0.*

A hybrid system α is asymptotically stable with respect to a point x_0 *if all traces $\sigma \in \tau(\alpha)$ are asymptotically stable w.r.t. x_0.*

In the following we assume w.l.o.g. that $x_0 = 0$.

Example 7. *Consider the following example of a dynamical system.*

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\frac{1}{2}x - \frac{1}{10}y \end{aligned}$$

As the system is given by linear differential equations we can analyze its behavior by looking at the eigenvalues of its defining matrix

$$\begin{pmatrix} 0 & 1 \\ -\frac{1}{2} & -\frac{1}{10} \end{pmatrix} \ .$$

The system is asymptotically stable w.r.t. $x = y = 0$ *as these eigenvalues are* $\frac{1}{20}(-1+i\sqrt{199})$ *and* $\frac{1}{20}(-1+i\sqrt{199})$ *which both have negative real parts (see e.g. [Kha96] for details on the behavior of linear systems). A plot of this system for initial values* $x = 10$, $y = 10$ *is given in Figure 3.6 for the* x*-*y*-plane and in Figure 3.7 for the* t*-*x*-plane. The system moves in a spiral motion towards the equilibrium* $x = y = 0$.

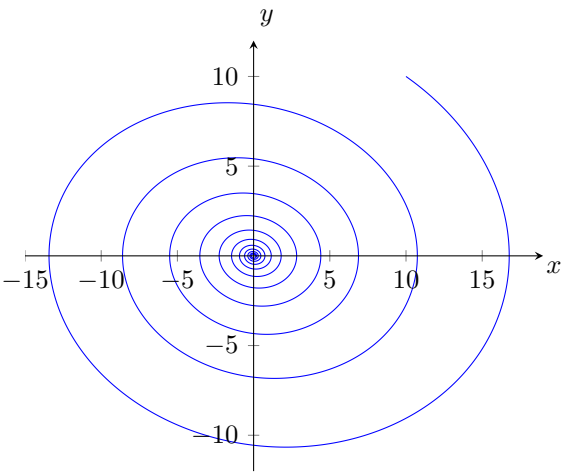


Figure 3.6: Example of a stable trajectory (x-y-plane)

Asymptotic stability is a property regarding the system behavior when time approaches infinity. It does not provide any guarantees about the convergence rate of the system towards its equilibrium. Therefore, we now recall the definition of exponential stability, a notion that is stronger than asymptotic stability in the sense that it guarantees exponential convergence towards the equilibrium.

Definition 28 (Exponential Stability [Kha96])**.** *If there exists two positive real numbers,* $k_1 > 0$ *and* $k_2 > 0$ *such that*

$$||\sigma(t)|| \leq k_1 e^{-k_2 t}||\sigma(0)||$$

for all $t \geq 0$ *then the hybrid trace* σ *is* exponentially stable w.r.t. the point $x_0 = 0$.

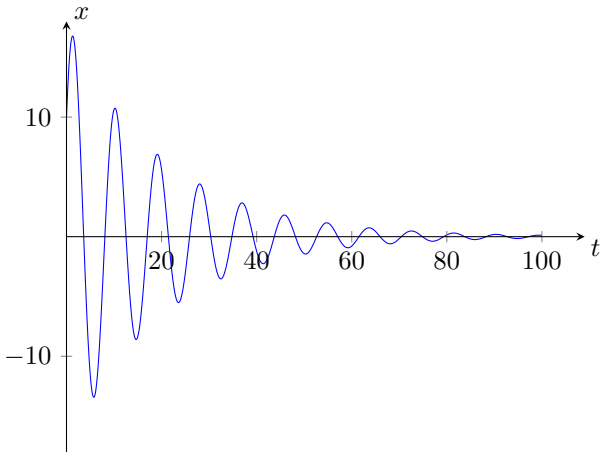


Figure 3.7: Example of a stable trajectory (t-x-plane)

A hybrid system α is exponentially stable *iff all traces $\sigma \in \tau(\alpha)$ are* exponentially stable w.r.t. $x_0 = 0$.

Observe that, exponential stability is a special case of asymptotic stability [Kha96].

Lemma 11. *If a trace σ is exponentially stable w.r.t. $x_0 = 0$ then it is also asymptotically stable w.r.t. $x_0 = 0$.*

Proof. See for example [Kha96]. □

From this lemma we can immediately draw the following corollary.

Corollary 1. *If a hybrid system α is exponentially stable, then it is asymptotically stable as well.*

The stability notions discussed so far share the fact that they define stability w.r.t. a single equilibrium point. Often, however, it is sufficient to know that the system will eventually stay within some region. Such a notion was introduced by Podelski and Wagner in 2006.

Definition 29 (Region Stability [PW06])**.** *A trajectory x is* region stable *w.r.t. a region R if there exists a point in time t_0 such that from then on, the trajectory stays within the region R, i.e.,*

$$\exists t_0 \forall t \geq t_0 \bullet x(t) \in R \ .$$

A hybrid system α is called region stable *iff all traces $\sigma \in \tau(\alpha)$ are region stable.*

Our similarity notion allows us to transfer stability properties between similar systems. A system that is a robust refinement of an asymptotically stable has to eventually stay in some region around the equilibrium. Therefore, we can consider this region stable in the sense of Podelski and Wagner.

In the following we are going to prove the relations sketched in Figure 3.8. Let $B_r \hat{=} \{x \mid x \in \mathbb{R}^n \wedge ||x|| < r\}$ denote the open n-dimensional ball with radius r centered at the origin.

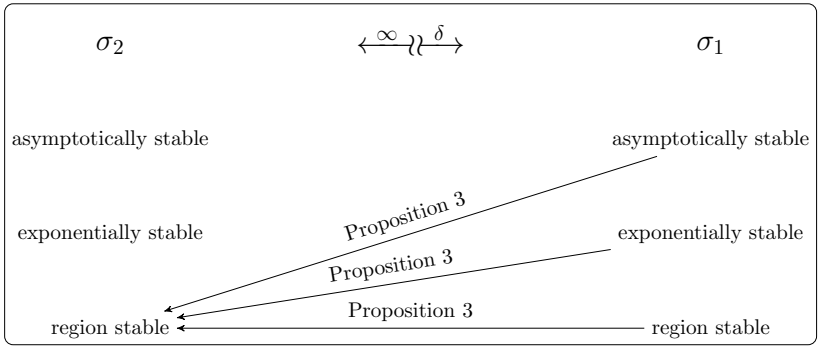


Figure 3.8: Weak similarity and stability

Proposition 3. *Let $\sigma_1 : \mathbb{R} \to \mathbb{R}^n$ and $\sigma_2 : \mathbb{R} \to \mathbb{R}^n$ be ε-δ-similar traces, i.e., $\sigma_1 \overset{\infty}{\longleftrightarrow}\!\!\!\!\!\!\sim\!\!\!\!\!\!\overset{\delta}{\longrightarrow} \sigma_2$.*

1. *If the trace σ_1 is asymptotically stable then for each $\gamma > 0$ the trace σ_2 is region stable w.r.t. the region $B_{\delta+\gamma}$.*
2. *If the trace σ_1 is exponentially stable then for every $\gamma > 0$ the trace σ_2 is region stable w.r.t. the region $B_{\delta+\gamma}$.*

3. *If the trace σ_1 is region stable w.r.t. some region R then the trace σ_2 is region stable w.r.t. the region S with*

$$S \hat{=} \{x | x \in \mathbb{R}^n \wedge \exists y \in R : ||x - y|| \leq \delta\} \ .$$

Proof. We start by proving the first statement. Let $\gamma > 0$ be arbitrary. Let δ_γ be such that if $||\sigma_1(0)|| < \delta_\gamma$ then for all times $t > 0$ we have that $||\sigma_1(t)|| < \gamma$.
As $\sigma_1 \overset{\infty}{\longleftrightarrow}\!\!\wr\wr\!\!\overset{\delta}{\longleftrightarrow} \sigma_2$ implies $\sigma_2 \overset{\infty}{\longleftrightarrow}\!\!\wr\wr\!\!\overset{\delta}{\longleftrightarrow} \sigma_1$ we know that there is an retiming $\mathfrak{r}$ such that

$$\forall (t, \tilde{t}) \in \mathfrak{r} : ||\sigma_2(t) - \sigma_1(\tilde{t})|| \leq \delta \ . \tag{3.6}$$

Let t_0 be such that $(t_0, 0) \in \mathfrak{r}$. We know that, for all $t_1 \geq 0$

$$||\sigma_1(t_1)|| < \gamma \ . \tag{3.7}$$

From (3.6) we get that for all t_2 with $(t_2, t_1) \in \mathfrak{r}$

$$||\sigma_2(t_2) - \sigma_1(t_1)|| \leq \delta \ . \tag{3.8}$$

Now we make a case distinction.

Case $||\sigma_2(t_2)|| \geq ||\sigma_1(t_1)||$**:** Using this assumption we know that

$$||\sigma_2(t_2)|| - ||\sigma_1(t_1)|| = |\, ||\sigma_2(t_2)|| - ||\sigma_1(t_1)||\, | \ .$$

From Lemma 2, we know that

$$|\, ||\sigma_2(t_2)|| - ||\sigma_1(t_1)||\, | \leq ||\sigma_2(t_2) - \sigma_1(t_1)|| \ .$$

Putting this together, using (3.8) and (3.7), we get that

$$||\sigma_2(t_2)|| \leq ||\sigma_2(t_2) - \sigma_1(t_1)|| + ||\sigma_1(t_1)|| < \delta + \gamma \ .$$

Case $||\sigma_2(t_2)|| < ||\sigma_1(t_1)||$**:** Using this assumption we know that

$$||\sigma_2(t_2)|| - ||\sigma_1(t_1)|| = |\, ||\sigma_1(t_1)|| - ||\sigma_2(t_2)||\, | \ .$$

From Lemma 2, we know that

$$|\, ||\sigma_1(t_1)|| - ||\sigma_2(t_2)||\, | \leq ||\sigma_1(t_1) - \sigma_2(t_2)|| \ .$$

However, using Lemma 1 we get that

$$||\sigma_1(t_1) - \sigma_2(t_2)|| = ||\sigma_2(t_2) - \sigma_1(t_1)|| \ .$$

Putting this together, using (3.8) and (3.7), we get that

$$||\sigma_2(t_2)|| \leq ||\sigma_2(t_2) - \sigma_1(t_1)|| + ||\sigma_1(t_1)|| < \delta + \gamma \ .$$

Overall, we get that $||\sigma_2(t_2)|| < \delta + \gamma$, and as $\mathfrak{r}$ is a retiming we know that this holds for all $t_2 \geq t_0$ which concludes the proof.

The second statement follows from Lemma 11. The proof for the third statement is analog. □

We can lift these results from hybrid traces to hybrid systems. This gives the following important theorem. We sketch the resulting relations in Figure 3.9.

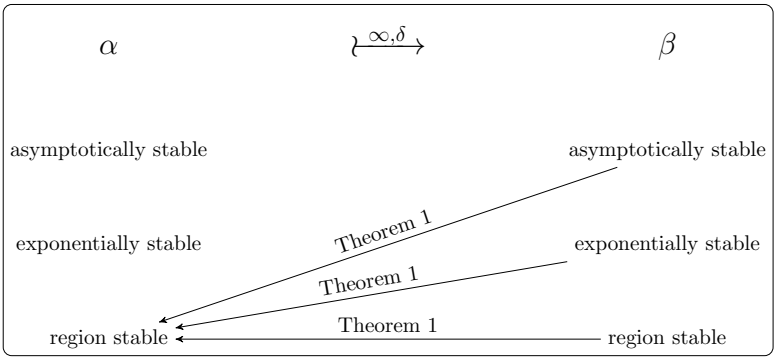


Figure 3.9: Weak robust refinement and stability

Theorem 1. *Let α and β be hybrid systems such that $\alpha \overset{\infty,\delta}{\rightarrowtail} \beta$.*

1. *If the system β is asymptotically stable then for every $\gamma > 0$ the system α is region stable w.r.t. the region $B_{\delta+\gamma}$.*

2. *If the system β is exponentially stable then for every $\gamma > 0$ the system α is region stable w.r.t. the region $B_{\delta+\gamma}$.*

3. *If the system β is region stable w.r.t. some region R then the system α is region stable w.r.t. the region S with*

$$S \hat{=} \{x | x \in \mathbb{R}^n \wedge \exists y \in R : ||x - y|| \leq \delta\} \ .$$

Observe that the contraposition of Theorem 1 provides a method for refuting a proposition that a system α is a weak robust refinement of another system β in the sense that $\alpha \leftsquigarrow^{\infty,\delta} \beta$. Since this is an important observation, we state it explicitly as an remark.

Remark 3. *If system β is asymptotically stable or exponentially stable and there is a trace $\sigma \in \tau(\alpha)$ and $\gamma > 0$ such that that σ is not region stable w.r.t. the region $B_{\delta+\gamma}$, then α cannot be a robust refinement of β.*

We are able to get an important corollary using Lemma 10 as we have proven the previous statements using our notion of weak δ-similarity (resp. weak δ-refinement) but they hold for ε-δ-similarity (resp. ε-δ-refinement) as well by Lemma 9 (resp. Lemma 10). This leads to the following corollary concerning traces. Again we sketch the resulting relations in Figure 3.10.

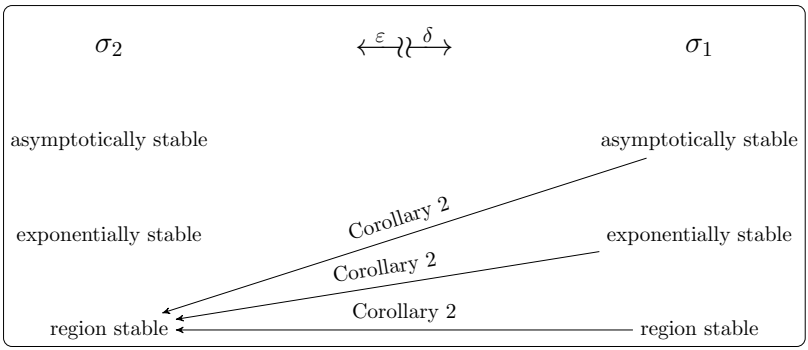


Figure 3.10: Similarity and stability

Corollary 2. *Let $\sigma_1 : \mathbb{R} \to \mathbb{R}^n$ and $\sigma_2 : \mathbb{R} \to \mathbb{R}^n$ be a trajectory such that $\sigma_1 \leftarrow \overset{\varepsilon}{\leftsquigarrow} \overset{\delta}{\rightarrow} \sigma_2$.*

1. *If the trace σ_1 is asymptotically stable then for every $\gamma > 0$ the trace σ_2 is region stable w.r.t. the region $B_{\delta+\gamma}$.*

2. *If the trace σ_1 is exponentially stable then for every $\gamma > 0$ the trace σ_2 is region stable w.r.t. the region $B_{\delta+\gamma}$.*

3. *If the trace σ_1 is region stable w.r.t. some region R then the trace σ_2 is region stable w.r.t. the region S with*

$$S \hat{=} \{x | x \in \mathbb{R}^n \wedge \exists y \in R : ||x - y|| \leq \delta\} \ .$$

In addition, we get the following corollary about hybrid systems (see Figure 3.11) by lifting the results from the previous corollary about hybrid traces.

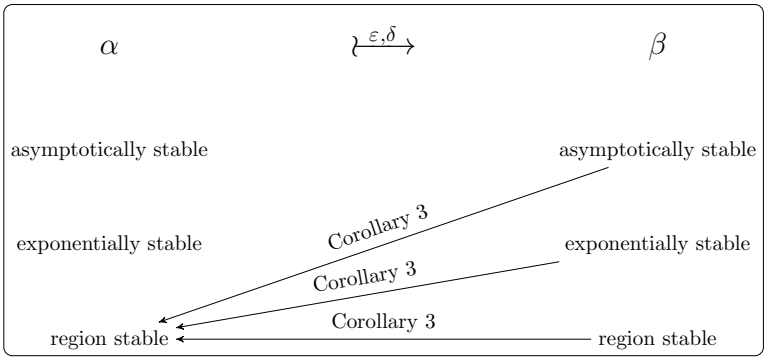


Figure 3.11: Robust refinement and stability

Corollary 3. *Let α, β by hybrid systems such that $\alpha \leadsto_{\varepsilon,\delta} \beta$*

1. *If the system β is asymptotically stable then for every $\gamma > 0$ the system α is region stable w.r.t. the region $B_{\delta+\gamma}$.*

2. *If the system β is exponentially stable then for every $\gamma > 0$ the system α is region stable w.r.t. the region $B_{\delta+\gamma}$.*

3. *If the system β is region stable w.r.t. some region R then the system α is region stable w.r.t. the region S with*

$$S \hat{=} \{x | x \in \mathbb{R}^n \wedge \exists y \in R : ||x - y|| \leq \delta\} \ .$$

Note that our notion of similarity does not guarantee that if a system is asymptotically stable that all its robust refinements are so as well. To illustrate this, consider the following example.

Example 8. *Let $x(t) = 0$ and $y(t) = \sin(t)$ for $t \in \mathbb{R}_{\geq 0}$ (see Figure 3.12). Obviously, $x \leftarrow\!\!\!\!\!\!\sim^{\infty}\!\!\!\!\!\!\sim^{\delta}\!\!\!\!\!\!\rightarrow y$ for $\delta = ||1||$. Thus, $x \leftarrow\!\!\!\!\!\!\sim^{\varepsilon}\!\!\!\!\!\!\sim^{\delta}\!\!\!\!\!\!\rightarrow y$ for all $\varepsilon \geq 0$ and $\delta \geq 1$. However, although x obviously satisfies the requirements for all our notions of stability the similar function y is neither exponentially nor asymptotically stable. Still, the function y will never leave any region $\{z \mid |z| \leq \delta + \gamma\}$ for $\gamma > 0$ and is, thus, region stable w.r.t. to those.*

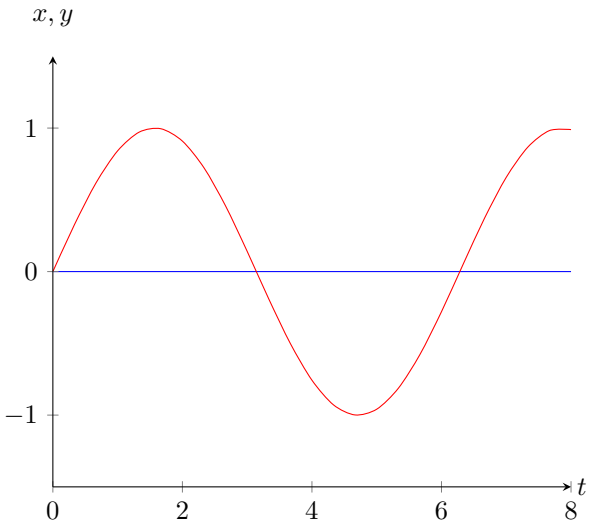


Figure 3.12: Example of a stable and a region stable trajectory

An interesting special case is if there are no spatial deviations. That is the case where for two traces σ_1 and σ_2 we either have $\sigma_1 \leftarrow\!\!\!\!\!\!\sim^{\infty}\!\!\!\!\!\!\sim^{0}\!\!\!\!\!\!\rightarrow \sigma_2$ or $\sigma_1 \leftarrow\!\!\!\!\!\!\sim^{\varepsilon}\!\!\!\!\!\!\sim^{0}\!\!\!\!\!\!\rightarrow \sigma_2$ for some fixed ε. An overview of the following results is provided in Figure 3.13 and Figure 3.14. For region stability we can derive two corollaries from our previous results. For asymptotic stability and exponential stability it gets even more interesting. We are now able to transfer properties of asymptotic stability as we are not allowing any

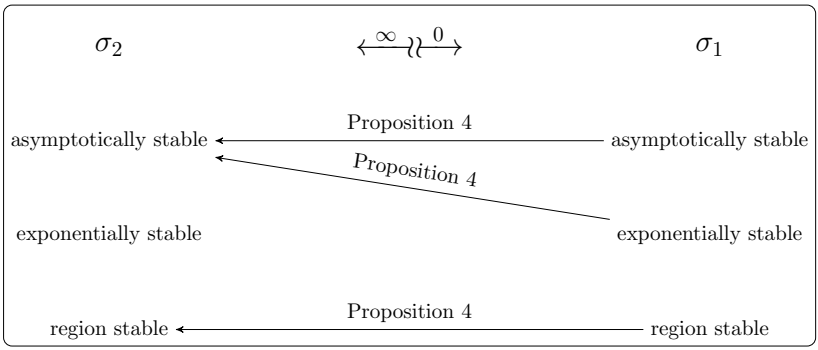


Figure 3.13: Weak 0-similarity and stability

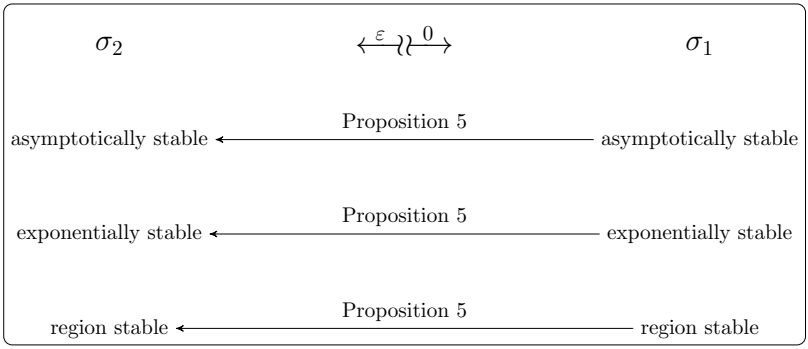


Figure 3.14: 0-Similarity and stability

spatial deviations and thus any similar trajectory is converging towards the origin as well.

Proposition 4. *Let $\sigma_1 : \mathbb{R} \to \mathbb{R}^n$, $\sigma_2 : \mathbb{R} \to \mathbb{R}^n$ be traces such that $\sigma_2 \leftarrow\!\!\!\!\!\!\sim^{\infty} \sim\!\!\!\!\!\!\rightarrow^{0} \sigma_1$.*

1. *If σ_1 is asymptotically stable then σ_2 is asymptotically stable as well.*
2. *If σ_1 is exponentially stable then σ_2 is asymptotically stable.*
3. *If σ_1 is region stable with respect to some region R then σ_2 is region stable with respect to the same region R.*

Proof. Assume σ_1 is asymptotically stable. In that case σ_2 is stable w.r.t. x_0 as well because the spatial distance to σ_1 is bounded by 0. In order to show that σ_2 is attracted to the point x_0 we use the property of the retiming $\mathfrak{r}$ witnessing $\sigma_2 \leftarrow\!\!\!\!\!\!\sim^{\infty} \sim\!\!\!\!\!\!\rightarrow^{0} \sigma_1$. This is, $\mathfrak{r}$ is surjective and therefore, for every $\tilde{t} \in \mathbb{R}_{\geq 0}$ we have that there is t such that $(t, \tilde{t}) \in \mathfrak{r}$. It is left-total and therefore for every $t \in \mathbb{R}_{\geq 0}$ there is $\tilde{t}$ such that $(t, \tilde{t}) \in \mathfrak{r}$. Furthermore, $\mathfrak{r}$ is order preserving, and thus σ_2 also converges towards x_0.

In case σ_1 is exponentially stable, we can derive from the first statement and Lemma 11 that σ_2 is asymptotically stable.

In case σ_1 is region stable to some region R we get from Theorem 1 and $\delta = 0$ that σ_2 is region stable w.r.t. the same region R. □

If in addition the temporal distance is bounded as well we can even transfer the property of exponential stability.

Proposition 5. *Let $\sigma_1 : \mathbb{R} \to \mathbb{R}^n$, $\sigma_2 : \mathbb{R} \to \mathbb{R}^n$ be traces such that $\sigma_2 \leftarrow\!\!\!\!\!\!\sim^{\varepsilon} \sim\!\!\!\!\!\!\rightarrow^{0} \sigma_1$ for some non-negative number $\varepsilon \in \mathbb{R}_{\geq 0}$.*

1. *If σ_1 is asymptotically stable then σ_2 is asymptotically stable as well.*
2. *If σ_1 is exponentially stable then σ_2 is asymptotically stable.*
3. *If σ_1 is exponentially stable then σ_2 is exponentially stable as well.*
4. *If σ_1 is region stable with respect to some region R then σ_2 is region stable with respect to the same region R.*

Proof. The first two and the fourth statement follow from Lemma 9 and Proposition 4.

In order to show that the third statement is valid we assume that σ_1 is exponentially stable. From the second statement we know that σ_2 is asymptotically stable. Therefore, we only have to show that there is an exponential function that provides bounds on the convergence rate. Let k_1, k_2 be such that $||\sigma_1(t)|| \leq k_1 e^{-k_2 t}||\sigma_1(0)||$. The worst case now is that the evolution of σ_2 is delayed by at most ε w.r.t. the evolution of σ_1. It is therefore sufficient to move the initial value of the exponential function up by e^{ε}. That way the value k_1 is reached ε time units later than before. Therefore, we get $||\sigma_2(t)|| \leq (k_1 + e^{\varepsilon})e^{-k_2 t}||\sigma_1(0)|| = (k_1 + e^{\varepsilon})e^{-k_2 t}||\sigma_2(0)||$. The latter equality holds by Lemma 4 and the fact that the spatial distance is at most 0. Hence σ_2 is exponentially stable. □

These results can be lifted from hybrid traces to hybrid systems again (see Figure 3.15 and Figure 3.16).

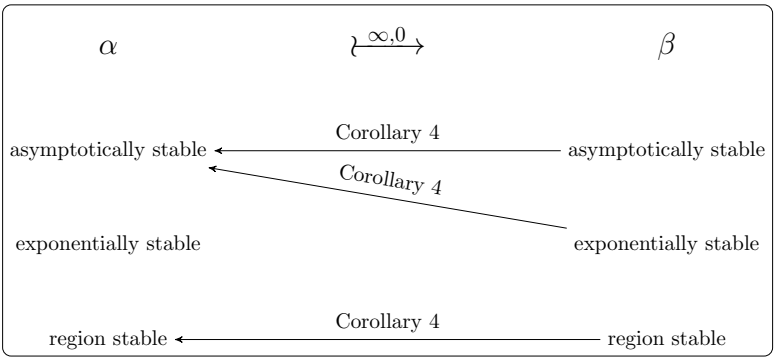


Figure 3.15: Weak 0-robust refinement and stability

Corollary 4. *Let α and β by hybrid systems such that $\alpha \rightsquigarrow_{\infty,0} \beta$.*

1. *If β is asymptotically stable then α is asymptotically stable as well.*
2. *If β is exponentially stable then α is asymptotically stable.*
3. *If β is region stable with respect to some region R then α is region stable with respect to the same region R.*

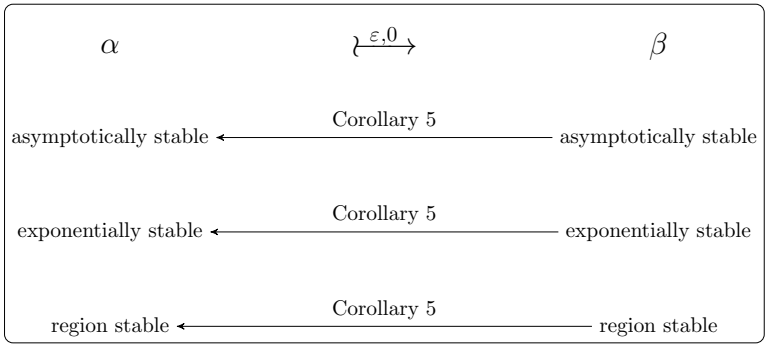


Figure 3.16: 0-robust refinement and stability

Corollary 5. *Let α and β by hybrid systems such that $\alpha \leadsto_{\varepsilon,0} \beta$ for some non-negative number $\varepsilon \in \mathbb{R}_{\geq 0}$.*

1. *If β is asymptotically stable then α is asymptotically stable as well.*
2. *If β is exponentially stable then α is asymptotically stable.*
3. *If β is exponentially stable then α is exponentially stable as well.*
4. *If β is region stable with respect to some region R then α is region stable with respect to the same region R.*

Observe that, for the cases where the spatial distance is 0, i.e., $\delta = 0$, the results from Proposition 3, Theorem 1, Corollary 2, and Corollary 3 of course are still valid. Hence for any type of stability we can conclude that the other object (trace or system) is region stable.

3.2.2 Linear Time Real-Time Temporal Logic

In addition to stability properties our notion of similarity can be used to transfer more complex properties between similar systems in a formal way. The quantitative nature of our similarity notion was chosen in order to use the bounds on the deviations between the system behaviors to syntactically calculate what properties are preserved. We thus give a syntactic transformation of formulas that are known to hold for one system into formulas that we know have to hold for all systems that are "close" w.r.t. certain

bounds. The idea is to use the upper bounds on the deviations to weaken the formulas to be sure that they still hold in all of robust refinements of a system satisfying the original formulas.

To express properties of real-time as well as hybrid systems a variety of different logics have been proposed in the literature (see [AH92] for a survey). We choose a variant of the future fragment of metric temporal logic (MTL) [Koy90] to specify properties of our systems. As basic propositions we use expressions that are evaluated on the system variables. This gives us a nice partitioning between temporal and spatial propositions on a syntactical level which we exploit when proving that certain properties are preserved by our refinement relation.

We call our real-valued real-time linear time temporal logic that we study in this section *natural logic* ($\mathcal{L}\natural$).

First, let us define what a Lipschitz continuous function is.

Definition 30 (Lipschitz continuity)**.** *We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is* Lipschitz continuous*, iff there is some constant M such that for all points $(x_1, \ldots, x_n) \in \mathbb{R}^n$, and all points $(y_1, \ldots, y_n) \in \mathbb{R}^n$ the following inequality holds:*

$$|f(x_1, \ldots, x_n) - f(y_1, \ldots, y_n)| \leq M \cdot ||(x_1, \ldots, x_n), (y_1, \ldots, y_n)|| \ .$$

We call the smallest M that has this property the Lipschitz constant *of f.*

We use this definition to restrict the basic terms in the syntax of our logic $\mathcal{L}\natural$.

Definition 31 (Syntax of $\mathcal{L}\natural$)**.** *The* syntax of natural logic formulas *is defined by the following grammar:*

$$\begin{aligned}\phi ::= &(x_1, \ldots, x_n) \in \mathcal{I} \mid f(x_1, \ldots, x_n) \leq 0 \mid g(x_1, \ldots, x_n) \leq 0 \\ &\mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \,\mathbb{U}_{\mathcal{J}}\, \phi_2\end{aligned}$$

where $\mathcal{I} \subseteq \mathbb{R}^n, \mathcal{J} \subseteq \mathbb{R}_{\geq 0}$, f is a Lipschitz continuous function, g is a function that is componentwise either monotonically increasing or monotonically decreasing, and x_i are variables.

Note that we distinguish between the cases of Lipschitz continuous functions f and monotone functions g on a syntactically level solely in order to make the proofs later on more readable.

We now turn our focus to the semantics of the logical formulas. The semantics of the basic terms imposes bounds on the valuations of the system variables. This can either be done by restricting the values of the variables to be within some n-dimensional real-valued set or by imposing a Lipschitz continuous or monotone constraint on multiple variables. Note that it is also fine to use only $k \leq n$ variables and a set $\mathcal{I} \subseteq \mathbb{R}^k$. The variables are connected to the system state using the valuation function given in Definition 20. Boolean connectives are interpreted as usual, and the "until"-operator provides us with the possibility to express both the temporal order of states as well as postulate time bounds on these orders. The set annotation forces the postcondition to hold at some point in time that lies within this set. However, as we consider finite in addition to infinite traces, we use a weak interpretation of the "until". That is, we consider such an "until"-formula satisfied if there is an extension to the current trace that satisfies the formula.

Definition 32 (Semantics)**.** *We define for a trace σ and some $t \in \mathbb{R}_{\geq 0}$ the* semantics of a formula ϕ *inductively as follows:*

If σ is defined at t in the sense of Definition 20 then:

$$\sigma, t \models (x_1, \ldots, x_n) \in \mathcal{I} \text{ iff } (\sigma(t)(x_1), \ldots, \sigma(t)(x_n)) \in \mathcal{I} \tag{3.9}$$

$$\sigma, t \models f(x_1, \ldots, x_n) \leq 0 \text{ iff } f(\sigma(t)(x_1), \ldots, \sigma(t)(x_n)) \leq 0 \tag{3.10}$$

$$\sigma, t \models g(x_1, \ldots, x_n) \leq 0 \text{ iff } g(\sigma(t)(x_1), \ldots, \sigma(t)(x_n)) \leq 0 \tag{3.11}$$

$$\sigma, t \models \neg\phi \text{ iff not } \sigma, t \models \phi \tag{3.12}$$

$$\sigma, t \models \phi \wedge \psi \text{ iff } \sigma, t \models \phi \text{ and } \sigma, t \models \psi \tag{3.13}$$

$$\sigma, t \models \phi \,\mathbb{U}_{\mathcal{J}}\, \psi \text{ iff } \exists t' \in \mathcal{J} : \sigma, t' + t \models \psi \text{ and } \forall t \leq t'' < t' + t : \sigma, t'' \models \phi \tag{3.14}$$

Otherwise, if σ is not defined at t in the sense of Definition 20 then all formulas are true.

Additionally, we define for a set of traces Σ:

$$\Sigma, t \models \phi \text{ iff for all traces } \sigma \in \Sigma \text{ holds } \sigma, t \models \phi \tag{3.15}$$

A hybrid system α satisfies a formula ϕ denoted by $\alpha \models \phi$, iff $\tau(\alpha), 0 \models \phi$.

We define some abbreviations:

$$\begin{aligned} true &\hat{=} 0 \leq 0 \\ false &\hat{=} \neg true \\ \phi \vee \psi &\hat{=} \neg(\neg\phi \wedge \neg\psi) \\ \phi \rightarrow \psi &\hat{=} \neg\phi \vee \psi \\ \phi \leftrightarrow \psi &\hat{=} (\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi) \end{aligned}$$

Furthermore, we define the *eventually* modality by

$$\Diamond_{\mathcal{J}} \phi \hat{=} true\, \mathbb{U}_{\mathcal{J}}\, \phi$$

and the *always* modality as the abbreviation

$$\Box_{\mathcal{J}} \phi \hat{=} \neg\Diamond_{\mathcal{J}} \neg\phi \ .$$

Also, we denote for a constant $a \in \mathbb{R}$ by $x \leq a \hat{=} x \in]-\infty, a]$, and by $x = a \hat{=} x - a \leq 0 \wedge a - x \leq 0$. Based on this, we can define $x > a \hat{=} \neg x \leq a$ and $x \geq a \hat{=} \neg x \leq a \vee x = a$. Using the same notion we can allow $\geq, =, <$ for comparisons with Lipschitz continuous or monotone functions as well.

Our notion of similarity can be seen as a decrease of the resolution of the image we have of the system behavior thus blurring the borders. If we originally knew that at some time between t and t' some event would happen, we now have to account for the timing deviations that might occur. Thus, if the event originally happened at time t it might now occur in the worst case already at $t-\varepsilon$. If it originally occurred at time t', the worst case we have to consider is that it now might occur as late as $t'+\varepsilon$. This widens the set of possible time points for the event, thus reducing our knowledge about exact timings. A similar effect happens on the variable valuations.

We define two syntactic transformation functions on formulas. The function $re_{\varepsilon,\delta}(\cdot)$ is applied if the current subformula is in a context of an even number of negations, whereas the function $ro_{\varepsilon,\delta}(\cdot)$ is applied to subformulas under an odd number of negations. Note that if the set indexing an until operator becomes empty, the formula is trivially false.

Definition 33 (Transformation Functions)**.** *Let $\mathcal{I} \subseteq \mathbb{R}^n, \mathcal{J} \subseteq \mathbb{R}_{\geq 0}$, $\lambda \geq 1$, and $||\cdot||_b$ be a monotone norm such that for all $x \in \mathbb{R}^n$ it holds that $\frac{1}{\lambda}||x|| \leq ||x||_b \leq \lambda||x||$. We define $re_{\varepsilon,\delta}(\cdot)$ inductively as the following function:*

- $re_{\varepsilon,\delta}((x_1,\ldots,x_n) \in \mathcal{I}) := (x_1,\ldots,x_n) \in \mathcal{I}'$, *where*

$$\mathcal{I}' = \{a \mid \exists b \in \mathcal{I} : ||b-a|| \leq \delta\} \ .$$

- $re_{\varepsilon,\delta}(f(x_1,\ldots,x_n) \leq 0) := f(x_1,\ldots,x_n) - \delta \cdot M \leq 0$ *where* M *is the Lipschitz constant for* f.

- $re_{\varepsilon,\delta}(g(x_1,\ldots,x_n) \leq 0) := g(e(x_1),\ldots,e(x_n)) \leq 0$ *where*

$$e(x_i) := \begin{cases} x_i - \lambda\frac{\delta}{||(\delta_{ij})_n||_b} & \text{iff } g \text{ is monotonically increasing in} \\ & \text{component } i \\ x_i + \lambda\frac{\delta}{||(\delta_{ij})_n||_b} & \text{otherwise} \end{cases} .$$

- $re_{\varepsilon,\delta}(\neg\phi) := \neg ro_{\varepsilon,\delta}(\phi)$.

- $re_{\varepsilon,\delta}(\phi \wedge \psi) := re_{\varepsilon,\delta}(\phi) \wedge re_{\varepsilon,\delta}(\psi)$.

- $re_{\varepsilon,\delta}(\phi \,\mathbb{U}_{\mathcal{J}}\, \psi) := re_{\varepsilon,\delta}(\phi)\, \mathbb{U}_{\mathcal{J}' \cap [0,\infty[}\, re_{\varepsilon,\delta}(\psi)$, *where*

$$\mathcal{J}' = \{a \mid \exists b \in \mathcal{J} : |b-a| \leq \delta\} \ .$$

The transformation function $ro_{\varepsilon,\delta}(\cdot)$ *is inductively defined by:*

- $ro_{\varepsilon,\delta}((x_1,\ldots,x_n) \in \mathcal{I}) := (x_1,\ldots,x_n) \in \mathcal{I}'$, *where*

$$\mathcal{I}' = \{a \mid \forall b \notin \mathcal{I} : ||b-a|| > \delta\} \ .$$

- $ro_{\varepsilon,\delta}(f(x_1,\ldots,x_n) \leq 0) := f(x_1,\ldots,x_n) + \delta \cdot M \leq 0$ *where* M *is the Lipschitz constant for* f.

- $ro_{\varepsilon,\delta}(g(x_1,\ldots,x_n) \leq 0) := g(o(x_1),\ldots,o(x_n)) \leq 0$ *where*

$$o(x_i) := \begin{cases} x_i + \lambda\frac{\delta}{||(\delta_{ij})_n||_b} & \text{iff } g \text{ is monotonically increasing in} \\ & \text{component } i \\ x_i - \lambda\frac{\delta}{||(\delta_{ij})_n||_b} & \text{otherwise} \end{cases} .$$

- $ro_{\varepsilon,\delta}(\neg\phi) := \neg re_{\varepsilon,\delta}(\phi)$.

- $ro_{\varepsilon,\delta}(\phi \wedge \psi) := ro_{\varepsilon,\delta}(\phi) \wedge ro_{\varepsilon,\delta}(\psi)$.

- $ro_{\varepsilon,\delta}(\phi \,\mathbb{U}_{\mathcal{J}}\, \psi) := ro_{\varepsilon,\delta}(\phi)\, \mathbb{U}_{\mathcal{J}'}\, ro_{\varepsilon,\delta}(\psi)$, *where*

$$\mathcal{J}' = \{a \mid \forall b \notin \mathcal{J} : |b - a| > \varepsilon\} \ .$$

Under an even number of negations the transformation function $re_{\varepsilon,\delta}(\cdot)$ widens all occurring sets. That is, if we have an expression of the form $x \in \mathcal{I}$ we enlarge the set $\mathcal{I}$ by adding the union of the δ-neighborhoods of all points within $\mathcal{I}$. For the "until"-operator we use the same construction using the ε-neighborhoods. See Figure 3.17 for an illustration. In the picture the balls around the points labeled with a have radius of δ. Note that only the balls on the border of the set $\mathcal{I}$ are necessary to consider. Observe that, the set $\mathcal{I}$ is not necessarily convex and therefore each point might be part of the boarder of $\mathcal{I}$. Even though the illustration is 2-dimensional the sets themselves are n-dimensional for the spatial and 1-dimensional for the temporal case. In the temporal case, we further intersect with the interval $[0, \infty[$ using the retiming property, i.e., the fact that a retiming is an order preserving relation. That way, the until operator still refers to events at this moment or in the future, rather than talking about events that might have happened in the past.

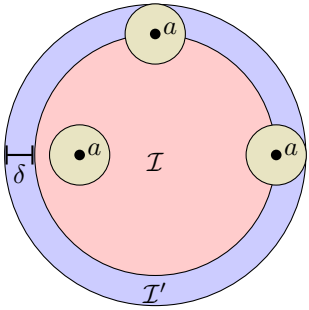


Figure 3.17: Enlarging a set by adding neighborhoods

Lipschitz continuous functions are moved towards negative values by subtracting δ times its Lipschitz constant. For component-wise monotone functions, we modify each component occurrence by moving it by a distance of δ w.r.t. to some monotone norm.

Under an odd number of negations, the transformations are basically doing the exact opposite. That is, the sets are contracted towards some core

by intersecting the set with all δ-neighborhoods (respectively ε-neighborhoods) of points that are not in the set. See Figure 3.18 for an illustration. In the picture the balls around the points labeled with a again have diameter δ. Like before, only the balls right outside the border of the set $\mathcal{I}$ are necessary to consider. Again, even though the illustration is 2-dimensional the sets themselves are n-dimensional for the spatial and 1-dimensional for the temporal case.

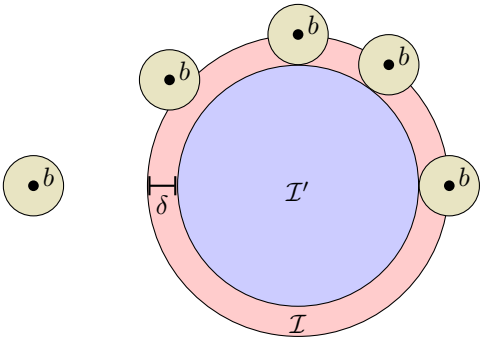


Figure 3.18: Contracting a set by subtracting neighborhoods

Observe that, the function $e(\cdot)$ and $o(\cdot)$ used in the monotone function case are doing basically the same just switching the cases. They are used to transform values of vector components w.r.t. a monotone norm $||\cdot||_b$. Recall that, $(\delta_{ij})_n$ denotes the vector that with component i being 1 and all others being 0. Further, observe that the existence of such a norm $||\cdot||_b$ and λ is guaranteed by Lemma 3 (see page 34). Additionally, if $||\cdot||$ is monotone then we can choose $\lambda = 1$ and $||\cdot||_b = ||\cdot||$. In case of the Euclidean or the maximum norm we further get that $||(\delta_{ij})_n||_b = 1$.

Remark 4. *As a side note, observe that if the sets used as annotations for the until operator are compact subsets of* $\mathbb{R}$*, i.e., intervals, we can calculate the transformation functions more easily:*

- $re_{\varepsilon,\delta}(\phi\, \mathbb{U}_{[a,b]}\, \psi) := re_{\varepsilon,\delta}(\phi)\, \mathbb{U}_{[a-\varepsilon,b+\varepsilon]}\, re_{\varepsilon,\delta}(\psi)$,
- $ro_{\varepsilon,\delta}(\phi\, \mathbb{U}_{[a,b]}\, \psi) := ro_{\varepsilon,\delta}(\phi)\, \mathbb{U}_{[a+\varepsilon,b-\varepsilon]}\, ro_{\varepsilon,\delta}(\psi)$

where $[x, y] := \emptyset$ *if* $x > y$.

Based on these definitions we can state an important theorem regarding our robust refinement notion.

Theorem 2. *If for two hybrid systems α and β it holds that $\alpha \leftsquigarrow^{\varepsilon,\delta} \beta$ and $\beta \models \phi$ then $\alpha \models re_{\varepsilon,\delta}(\phi)$.*

In order to prove this theorem we prove a slightly stronger lemma at the level of hybrid traces.

Lemma 12. *For two traces σ_A and σ_B if $\sigma_A \leftarrow^{\varepsilon} \rightsquigarrow^{\delta} \sigma_B$ then for all times $t \in \mathbb{R}_{\geq 0}$ if $\sigma_B, t \models \phi$ then $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi)$ for all t' with $(t', t) \in \mathfrak{r}$ where $\mathfrak{r}$ is a retiming witnessing $\sigma_A \leftarrow^{\varepsilon} \rightsquigarrow^{\delta} \sigma_B$.*

We apply structural induction to prove this lemma. The fact that the spatial distance is bounded can be used to discharge the base cases of the induction. The idea is that if the formula restricts the valuations to a set of points in $\mathbb{R}^n$ then the union of all δ-balls around each of these points contains all possible values for all systems that robustly refine it. A similar idea is applied for negations. There we have to shrink the region where we are sure that our values are not such that all δ-balls are contained within this region.

For the until-operator we can use the fact that the temporal distance is bounded by ε. Therefore, similar points can be found with ε-environments around the original times of the event.

Proof of Lemma 12. We prove Lemma 12 by structural induction. Let $t \in \mathbb{R}_{\geq 0}$ be arbitrary, $\sigma_A \leftarrow^{\varepsilon} \rightsquigarrow^{\delta} \sigma_B$, and $\sigma_B, t \models \phi$. Further, let $\mathfrak{r}$ be a retiming witnessing $\sigma_A \leftarrow^{\varepsilon} \rightsquigarrow^{\delta} \sigma_B$.

Base cases.

1. Case $\phi \equiv (x_1, \ldots, x_n) \in \mathcal{I}$

   Using Definition 33 we get that

   $$re_{\varepsilon,\delta}((x_1, \ldots, x_n) \in \mathcal{I}) = (x_1, \ldots, x_n) \in \mathcal{I}'$$

   with $\mathcal{I}' = \{a \mid \exists b \in \mathcal{I} : ||b - a|| \leq \delta\}$. From $\sigma_A \leftarrow^{\varepsilon} \rightsquigarrow^{\delta} \sigma_B$ we know that

   $$\forall (t, t') \in \mathfrak{r} \bullet ||\sigma_A(t) - \sigma_B(t')|| \leq \delta \ . \tag{3.16}$$

Let $b = (\sigma_A(t)(x_1), \ldots, \sigma_B(t)(x_n))$. From (3.16) we know that for all t' with $(t', t) \in \mathfrak{r}$ it holds that $||\sigma_A(t') - b|| \leq \delta$. Thus,

$$\sigma_A, t' \models (x_1, \ldots, x_n) \in \mathcal{I}' \ .$$

2. Case $\phi \equiv f(x_1, \ldots, x_n) \leq 0$

   On the same line as for the first case:

   In this case $re_{\delta,\varepsilon}\phi \equiv f(x_1, \ldots, x_n) - \delta \cdot M \leq 0$. $\sigma_A \overset{\varepsilon}{\leftarrow\!\wr\!\rightarrow}{}^{\delta} \sigma_B$ means that

   $$\forall (t, \tilde{t}) \in \mathfrak{r} \bullet ||\sigma_A(t) - \sigma_B(\tilde{t})|| \leq \delta \ .$$

   As $\sigma_B, t \models \phi$ we know that

   $$f(\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_n)) \leq 0$$

   As f is Lipschitz continuous we can conclude that for all points y with

   $$||y - \sigma_B(t)|| \leq d$$

   it holds that $f(y) - d \cdot M \leq 0$ where M is the Lipschitz constant of f.

   Therefore, for all t' with $(t', t) \in \mathfrak{r}$ we know that

   $$f(\sigma_A(t')(x_1), \ldots, \sigma_A(t')(x_n)) - \delta \cdot M \leq 0 \ .$$

   And, thus, $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi)$.

3. Case $\phi \equiv g(x_1, \ldots, x_n) \leq 0$

   In this case $re_{\varepsilon,\delta}(\phi) \equiv g(e(x_1), \ldots, e(x_n)) \leq 0$ where

   $$e(x_i) := \begin{cases} x_i - \lambda \frac{\delta}{||(\delta_{ij})_n||_b} & \text{iff } g \text{ is monotonically increasing in} \\ & \text{component } i \\ x_i + \lambda \frac{\delta}{||(\delta_{ij})_n||_b} & \text{otherwise} \end{cases} .$$

   We know that

   $$\forall (t, \tilde{t}) \in \mathfrak{r} \bullet ||\sigma_A(t) - \sigma_B(\tilde{t})|| \leq \delta \ .$$

   As $\sigma_B, t \models \phi$ we further know that

   $$g(\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_n)) \leq 0$$

For components i where g is monotonically increasing we know that for all $d \leq x_i$ it holds that $g(x_1, \ldots, d, \ldots, x_n) \leq g(x_1, \ldots, x_i, \ldots, x_n)$. For components i where g is monotonically decreasing we know that for all $d \geq x_i$ it holds that $g(x_1, \ldots, d, \ldots, x_n) \leq g(x_1, \ldots, x_i, \ldots, x_n)$.

The distance with respect to the norm $||\cdot||$ between the points

$$(\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_i))$$

and

$$(\sigma_A(t')(x_1), \ldots, \sigma_A(t')(x_i))$$

is bounded by δ. Therefore, $\lambda\delta$ is a bound on the distance w.r.t. $||\cdot||_b$. As the norm $||\cdot||_b$ is monotone, we can conclude from (2.13) (see page 32) that for $i \in \{1, \ldots, n\}$ it holds that

$$|\sigma_A(t')(x_i) - \sigma_B(t)(x_i)| \leq \lambda \frac{\delta}{(\delta_{ij})_n} \ .$$

That means, however, that $\sigma_A(t')(x_i) - \lambda \frac{\delta}{(\delta_{ij})_n} \leq \sigma_B(t)(x_i)$. Also

$$\sigma_A(t')(x_i) + \lambda \frac{\delta}{(\delta_{ij})_n} \geq \sigma_B(t)(x_i) \ .$$

Using the componentwise monotonicity of g, we can conclude that for all t' with $(t', t) \in \mathfrak{r}$ it holds that

$$g(e(\sigma_A(t')(x_1)), \ldots, e(\sigma_A(t')(x_n))) \leq 0 \ .$$

And, thus, $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi)$.

Induction hypothesis (IH). The property already holds for all formulas that are structurally simpler. That is if $\sigma_B, t \models \phi$ then $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi)$ for all t' with $(t', t) \in \mathfrak{r}$.

Induction step.

1. Case $\phi \equiv \neg\phi_1$

   We prove this case by induction over the structure of ϕ_1.

   - Base steps:

a) Case $\phi \equiv \neg(x_1, \ldots, x_n) \in \mathcal{I}$
In this case

$$\begin{aligned} re_{\varepsilon,\delta}(\phi) &\equiv \neg ro_{\varepsilon,\delta}((x_1, \ldots, x_n) \in \mathcal{I}) \\ &\equiv \neg(x_1, \ldots, x_n) \in \{a \mid \forall b \notin \mathcal{I} ||b-a|| \leq \delta\} \ . \end{aligned}$$

As we know that $\sigma_B, t \models \phi$ we can conclude that

$$\sigma_B, t \not\models (x_1, \ldots, x_n) \in \mathcal{I} \ .$$

From $\sigma_A \xleftarrow{\varepsilon}\!\rightsquigarrow\!\xrightarrow{\delta} \sigma_B$ we know that for all t' with $(t', t) \in \mathfrak{r}$ it holds that

$$\begin{aligned} &||(\sigma_A(t')(x_1), \ldots, \sigma_A(t')(x_n)) - \\ &\qquad (\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_n))|| \leq \delta \ . \end{aligned}$$

That means that

$$(\sigma_A(t')(x_1), \ldots, \sigma_A(t')(x_n)) \notin \mathcal{I} \backslash \{a \mid \exists b \notin \mathcal{I} : ||b - a|| \leq \delta\} \ .$$

This is

$$(\sigma_A(t')(x_1), \ldots, \sigma_A(t')(x_n)) \notin \{a \mid \forall b \notin \mathcal{I} : ||b-a|| > \delta\} \ .$$

Thus, $\sigma_A, t' \models \phi$ for all t' with $(t', t) \in \mathfrak{r}$.

b) Case $\phi \equiv \neg f(x_1, \ldots, x_n) \leq 0$
In this case

$$\begin{aligned} re_{\varepsilon,\delta}(\phi) &\equiv \neg ro_{\varepsilon,\delta}(f(x_1, \ldots, x_n) \leq 0) \\ &\equiv \neg f(x_1, \ldots, x_n) + \delta \cdot M \leq 0 \ . \end{aligned}$$

As $\sigma_B, t \models \phi$, we know that $\sigma_B, t \not\models f(x_1, \ldots, x_n) \leq 0$. Thus, we have that $f(\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_n)) > \delta$. Further, from $\sigma_A \xleftarrow{\varepsilon}\!\rightsquigarrow\!\xrightarrow{\delta} \sigma_B$ we can conclude that for all t' with $(t', t) \in \mathfrak{r}$ it holds that

$$||\sigma_A(t') - \sigma_B(t)|| \leq \delta \ .$$

As f is Lipschitz continuous with Lipschitz constant M we know that for all points y with $||y - \sigma_B(t)|| \leq \delta$ it holds that $f(y) + \delta \cdot M > 0$. Thus, $\sigma_A, t' \not\models f(x_1, \ldots, x_n) + \delta \cdot M \leq 0$ and, therefore, for all t' with $(t', t) \in \mathfrak{r}$ we can conclude that $\sigma_A, t' \models \phi$.

c) Case $\phi \equiv \neg g(x_1, \ldots, x_n) \leq 0$

This means $g(\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_n)) > 0$.

In this case $ro_{\varepsilon,\delta}(\phi) \equiv g(o(x_1), \ldots, o(x_n)) \leq 0$ where

$$o(x_i) := \begin{cases} x_i + \lambda \frac{\delta}{||(\delta_{ij})_n||_b} & \text{iff } g \text{ is monotonically} \\ & \text{increasing in component } i \\ x_i - \lambda \frac{\delta}{||(\delta_{ij})_n||_b} & \text{otherwise} \end{cases} .$$

$\sigma_A \overset{\varepsilon}{\leftrightsquigarrow}{}^{\delta} \sigma_B$ means that

$$\forall (t, \tilde{t}) \in \mathfrak{r} \bullet ||\sigma_A(t) - \sigma_B(\tilde{t})|| \leq \delta \ .$$

As $\sigma_B, t \models \phi$ we know that

$$g(\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_n)) \leq 0$$

For components i where g is monotonically increasing we know that for all $d \geq x_i$ it holds that

$$g(x_1, \ldots, d, \ldots, x_n) \geq g(x_1, \ldots, x_i, \ldots, x_n) \ .$$

For components i where g is monotonically decreasing we know that for all $d \leq x_i$ again we have that

$$g(x_1, \ldots, d, \ldots, x_n) \geq g(x_1, \ldots, x_i, \ldots, x_n) \ .$$

The distance between the points $(\sigma_B(t)(x_1), \ldots, \sigma_B(t)(x_i))$ and $(\sigma_A(t')(x_1), \ldots, \sigma_A(t')(x_i))$ w.r.t. the norm $||\cdot||$ is bounded by δ. Therefore, $\lambda\delta$ is a bound on the distance w.r.t. $||\cdot||_b$. As the norm $||\cdot||_b$ is monotone, we can conclude from (2.13) (see page 32) that for $i \in \{1, \ldots, n\}$ it holds that

$$|\sigma_A(t')(x_i) - \sigma_B(t)(x_i)| \leq \lambda \frac{\delta}{(\delta_{ij})_n} \ .$$

That means, however, that $\sigma_A(t')(x_i) + \lambda \frac{\delta}{(\delta_{ij})_n} \geq \sigma_B(t)(x_i)$. Also $\sigma_A(t')(x_i) - \lambda \frac{\delta}{(\delta_{ij})_n} \leq \sigma_B(t)(x_i)$.

Using the componentwise monotonicity of g, we can conclude that for all t' with $(t', t) \in \mathfrak{r}$ it holds that

$$g(o(\sigma_A(t')(x_1)), \ldots, o(\sigma_A(t')(x_n))) > 0 \ .$$

Therefore, it does not hold that

$$g(o(\sigma_A(t')(x_1)), \ldots, o(\sigma_A(t')(x_n))) \leq 0 \ .$$

And, thus, $\sigma_A, t' \models ro_{\varepsilon,\delta}(\phi)$.

- Induction hypothesis (IH2): The property already holds for all formulas that are structurally simpler. That is, if $\sigma_B, t \not\models \phi_1$ then $\sigma_A, t' \not\models ro_{\varepsilon,\delta}(\phi_1)$ for all t' with $(t', t) \in \mathfrak{r}$.
- Induction step:
  a) Case $\phi \equiv \neg\neg\phi_1$

  Resolving the double negation leads to $\phi \equiv \phi_1$ and for ϕ_1 we know by the outer induction hypothesis (IH) that the property is satisfied.

  b) Case $\phi \equiv \neg\,(\phi_1 \wedge \phi_2)$

  As $\sigma_B, t \models \phi$, we know that $\sigma_B, t \not\models \phi_1 \wedge \phi_2$. This is the case if $\sigma_B, t \not\models \phi_1$ or if $\sigma_B, t \not\models \phi_2$. Now we make a case distinction:

  Case 1 Assume $\sigma_B, t \models \neg\phi_1$. In this case we know by induction hypothesis (IH2) that $\sigma_A, t' \not\models ro_{\varepsilon,\delta}(\phi_1)$ for all t' with $(t', t) \in \mathfrak{r}$ and therefore

  $$\sigma_A, t' \not\models ro_{\varepsilon,\delta}(\phi_1) \wedge ro_{\varepsilon,\delta}(\phi_2) \ .$$

  Case 2 Assume $\sigma_B, t \models \neg\phi_2$. In this case we know by induction hypothesis (IH2) that $\sigma_A, t' \not\models ro_{\varepsilon,\delta}(\phi_2)$ for all t' with $(t', t) \in \mathfrak{r}$ and therefore

  $$\sigma_A, t' \not\models ro_{\varepsilon,\delta}(\phi_1) \wedge ro_{\varepsilon,\delta}(\phi_2) \ .$$

  However, this, by definition of $ro_{\varepsilon,\delta}(\cdot)$, is equivalent to

  $$\sigma_A, t' \not\models ro_{\varepsilon,\delta}(\phi_1 \wedge \phi_2) \ .$$

  And, thus, we can conclude that

  $$\sigma_A, t' \models \neg ro_{\varepsilon,\delta}(\phi_1 \wedge \phi_2) \ .$$

c) Case $\phi \equiv \neg(\phi_1 \,\mathbb{U}_{\mathcal{J}'}\, \phi_2)$

Let t' be such that $(t', t) \in \mathfrak{r}$. Further, let $\tilde{t} \geq t'$ be such that $\tilde{t} - t' \in \mathcal{J}' = \{a \mid \forall b \notin \mathcal{J} : |b - a| > \varepsilon\}$. It is obvious that $s - t \in \mathcal{J}$ for each s with $(\tilde{t}, s) \in \mathfrak{r}$. Therefore, we know that for all $\tilde{s}$ with $(\tilde{t}, \tilde{s}) \in \mathfrak{r}$ it holds that there is $t \leq t'' < \tilde{s}$ with $\sigma_B, t'' \not\models \phi_1$ or $\sigma_B, \tilde{s} \not\models \phi_2$. By IH2 this gives that for all s'' with $(s'', t'') \in \mathfrak{r}$: $\sigma_A, s'' \not\models ro_{\varepsilon,\delta}(\phi_1)$ or $\sigma_A, \tilde{t} \not\models ro_{\varepsilon,\delta}(\phi_2)$. Thus, $\sigma_A, t' \not\models ro_{\varepsilon,\delta}(\phi_1)\, \mathbb{U}_{\mathcal{J}'}\, ro_{\varepsilon,\delta}(\phi_2)$. This, however is equivalent to the statement

$$\sigma_A, t' \models \neg ro_{\varepsilon,\delta}(\phi_1 \,\mathbb{U}_{\mathcal{J}}\, \phi_2) \ .$$

2. Case $\phi \equiv \phi_1 \wedge \phi_2$:

If $\sigma_B, t \models \phi_1 \wedge \phi_2$ then $\sigma_B, t \models \phi_1$ and $\sigma_B, t \models \phi_2$. By induction hypothesis that means that for all t' with $(t', t) \in \mathfrak{r}$ $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi_1)$ and $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi_2)$, and thus $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi_1) \wedge re_{\varepsilon,\delta}(\phi_2)$. Therefore, $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi)$.

3. Case $\phi \equiv \phi_1 \,\mathbb{U}_{\mathcal{J}}\, \phi_2$

Therefore $\sigma_B, t \models \phi_1 \,\mathbb{U}_{\mathcal{J}}\, \phi_2$, which means that there is $\tilde{t} \in \mathcal{J}$ such that for all $t \leq t'' < \tilde{t} + t$ it holds that $\sigma_B, t'' \models \phi_1$ and $\sigma_B, \tilde{t} + t \models \phi_2$. By induction hypothesis IH we can conclude that for all s'' with $(s'', t'') \in \mathfrak{r}$ it holds that $\sigma_A, s'' \models re_{\varepsilon,\delta}(\phi_1)$. Further, for all $\tilde{s}$ with $(\tilde{s}, \tilde{t}) \in \mathfrak{r}$ we get that $\sigma_A, \tilde{s} \models re_{\varepsilon,\delta}(\phi_2)$. As the distance between s'' and t'' is bounded by ε and the distance between $\tilde{s}$ and $\tilde{t}$ is so as well, we can conclude that they are contained in

$$\mathcal{J}' := \{a \mid \exists b \in \mathcal{J} : |b - a| \leq \varepsilon\} \ .$$

Further, we have that $\mathfrak{r}$ is a retiming. Hence it is an order preserving relation. Assume that $\tilde{s} < t'$. We know that $(\tilde{s}, \tilde{t} + t) \in \mathfrak{r}$ and $(t', t) \in \mathfrak{r}$. From $t' > \tilde{s}$ and the fact that $\mathfrak{r}$ is order preserving we can conclude that $t \geq \tilde{t} + t$. Therefore, $\tilde{t} = 0$. In that case, we know that $\sigma_B, t \models \phi_2$ and get using our induction hypothesis IH that $\sigma_A, t' \models re_{\varepsilon,\delta}(\phi_2)$. Therefore, we can assume that $\tilde{s} \geq t'$ for the until operator, as otherwise the post condition is already satisfied. Hence we can intersect $\mathcal{J}'$ with $[0, \infty[$.

Overall, for t' with $(t', t) \in \mathfrak{r}$ we get

$$\sigma_A, t' \models re_{\varepsilon,\delta}(\phi_1)\, \mathbb{U}_{\mathcal{J}' \cap [0,\infty[}\, re_{\varepsilon,\delta}(\phi_2) \ . \qquad \square$$

Using this lemma we can now prove Theorem 2.

Proof of Theorem 2. Assume $\alpha \wr^{\varepsilon,\delta}\!\rightarrow \beta$, and $\beta \models \phi$.

Let $\sigma_A \in \tau(\alpha)$ be arbitrary. As $\alpha \wr^{\varepsilon,\delta}\!\rightarrow \beta$ we know that there is some $\sigma_B \in \tau(\beta)$ such that $\sigma_A \leftarrow\!\wr^{\varepsilon}_{\delta}\!\rightarrow \sigma_B$. Further, let $\mathfrak{r}$ be a retiming witnessing $\sigma_A \leftarrow\!\wr^{\varepsilon}_{\delta}\!\rightarrow \sigma_B$. As $\beta \models \phi$ we know that $\sigma_B, 0 \models \phi$. From Lemma 12 we can thus conclude that for all $(t, 0) \in \mathfrak{r}$ it holds that $\sigma_A, t \models re_{\varepsilon,\delta}(\phi)$. From Lemma 4 we know that $(0, 0) \in \mathfrak{r}$. Therefore, $\sigma_A, 0 \models re_{\varepsilon,\delta}(\phi)$. As σ_A was arbitrary this holds for all $\sigma_A \in \tau(\alpha)$. Thus $\tau(\alpha), 0 \models re_{\varepsilon,\delta}(\phi)$ which means that $\alpha \models re_{\varepsilon,\delta}(\phi)$. □

To establish Theorem 2, we decided to restrict our logic to Lipschitz continuous and monotone functions. We now illustrate the effect of functions h that have neither of these properties.

Example 9. *Consider two trajectories $x_1(t) = \frac{\delta}{2}$ and $x_2(t) = 0$. Then it is obvious that $x_1 \leftarrow^{0}\wr^{||\delta||}\rightarrow x_2$ holds. Now we examine what happens to the property described by $h(x_2) \leq 0$ where $h(z) := z^2$. The only value for which this holds is $x_2 = 0$. However, we cannot determine this from the spatial bounds of our similarity notion, that is $(x_1 - ||\delta||)^2 \leq 0$ as well as $(x_1 + ||\delta||)^2 \leq 0$ are false. The only thing we could preserve is the property that there is $\delta' \in [-||\delta||, ||\delta||]$ such that $(x_1 + \delta')^2 \leq 0$. However, adding quantifiers would make non-local transformations necessary and, therefore, not serve our goal of providing a simple transformation function that can be applied on the resulting properties again if necessary*

Using the weaker versions of similarity and refinement we can also transfer some properties as already discussed in details for stability properties. The version is weaker with respect to knowledge about timing. Those timings are only present in the intervals indexing the until-operators in the formulas. Thus weakening these operators by replacing those intervals by $[0, +\infty[$ removes all exact timing informations. Only temporal properties are preserved in that case, i.e., we still retain knowledge about event orders.

Theorem 3. *If for hybrid systems α and β both $\alpha \wr^{\infty,\delta}\!\rightarrow \beta$ and $\beta \models \phi$ hold, where ϕ does not contain any until-operations in a negative context, then $\alpha \models \phi^{+\delta}_{\sim}$ where $\phi^{+\delta}_{\sim} b = w(re_{0,\delta}(\phi))$ and w replaces the index of every until operator by $\mathbb{R}_{\geq 0}$.*

The proof follows easily from the proof for Theorem 2 by altering the induction steps for the until operator. Unfortunately, we lose too much information about the timings to keep knowledge about until operations in a negative context, i.e., under an odd number of negations. Hence the restriction to positive contexts in this theorem is necessary.

Note that we can use the functions $re_{\varepsilon,\delta}(\cdot)$ and $ro_{\varepsilon,\delta}(\cdot)$ to compute which properties a specification should have, in order to ensure that all its robust refinements satisfy a certain property ϕ.

Remark 5. *Let β be a hybrid system. If $\beta \models ro_{\varepsilon,\delta}(\phi)$ then for all hybrid systems α with $\alpha \leftarrowtail^{\varepsilon,\delta} \beta$ we know that $\alpha \models \phi$.*

Furthermore, observe that in case of ε-0-refinement we can allow for arbitrary functions instead of restricting the syntax to Lipschitz-continuous and componenetwise-monotone functions. This follows from the fact that we have an identity relation connecting on states in the special case where $\delta = 0$ holds.

3.3 Related Work

The idea to study similarity is not new and has been done in the past for several classes of systems.

Weighted or priced timed automata can be used to specify limited classes of hybrid systems. These systems feature two distinct sets of variables. There are clocks that can only be reset to 0 and evolve with a constant rate of 1 in each mode. These clocks can be used to formulate guards on the system transitions. Furthermore, there are weights. Weights evolve along constant differential equations and can be increased or decreased at transition by some constant. Thrane et al. [TFL10] studied different types of simulations that allow for some deviations of the costs in weighted timed automata. That is, they introduced a notion of similarity that allows to consider systems as similar even though there is a bounded deviation in the values of the prices.

Tabuada [Tab09] surveys equivalence and exact refinement notions for hybrid systems. He presents a game based approach to showing that two systems are in refinement relation and for the more general control problem, i.e., for showing that there is a controller restricting one system in such a way that it is a refinement of the other. The notions of approximate simulation presented in his book are taken from the work of Girard

et al. [GJP08]. They present a notion of approximate simulation that allows us to consider hybrid systems with identical control graphs as similar if there is only a bounded deviation between the variable values at each point in time. Thus, this notion strongly relates to the case of 0-δ-refinement relations. However, they consider simulation instead of refinement. Therefore, they demand a strong coupling of states of the implementation and the specification, whereas our notion is based on the complete runs of the systems. In case of deterministic systems their notion coincides with the special case of 0-δ-refinement.

The most important difference to our notion of similarity is that the notion of ε-δ-refinement allows us to consider systems as similar, where the actual distance is increasing unboundedly as Example 10 illustrates.

Example 10. *Consider two instances of the continuous system described by the differential equation system $\dot{x} = x$. The initial valuation of the variable x determines the initial slope. However, every positive initial valuation will result in a qualitative similar trajectory. Consider two instances with initial values a and b, where $a < b$. Then the initial distance is bounded by $||b - a||$ and we can find ε such that the two systems are ε-$||b - a||$-similar. Still, there is no δ such that the systems are in 0-δ-refinement relation. The issue with increasing spatial distance is illustrated on an example in Figure 3.19.*

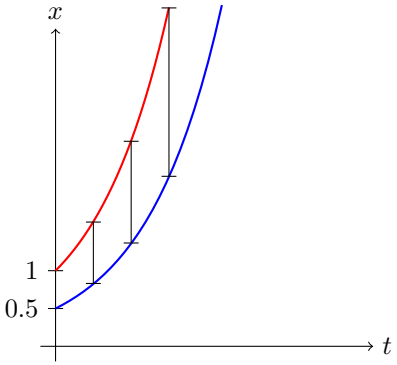


Figure 3.19: Plot of two exponential functions ($\dot{x} = x$) with initial values $x = 1$ and $x = \frac{1}{2}$.

Note that the same effect already occurs for polynomial functions with a degree of at least 2. This is, the effect is not limited to the case of exponential functions.

Other effects that are not covered by 0-δ-refinement are effects like sampling that often occur in implementations influence the timing behavior similarity notions have been studied that allow for some deviation in the timing behavior as well. Davoren [Dav09] presented an approach generalizing Skorokhod-metrics on hybrid traces and provided conditions under which these initially pseudo-metrics induce topologies which are Hausdorff and are thus indeed metrics. However, she does not present a constructive method for determining the values assigned to two concrete traces by the metric nor does she study the implications for sets of traces, i.e., hybrid systems.

Our notion of similarity is strongly inspired by that of Girard, Julius, and Pappas [GJP08] and the more general similarity notion in Davoren's work [Dav09]. That is, we study a refinement relation that allows one system to simulate another in a robust way, where deviations in continuous variable valuations, and in timing behavior are subject to constant bounds. However, we drop the requirement inherent to the work in [GJP08] that the discrete behavior must have the same control graph w.r.t. the locations of the automata.

Henzinger et al. [HMP05] study similarities between *timed systems*. As timed systems do not have variables other than clocks, they define their notion of similarity based on event orders. That is, two systems are similar if for every trace of one system there is a trace of the other that contains the same number and order of events but may differ in the event timings. Now they take the maximum timing difference in order to get a quantitative notion of similarity. Furthermore, they link their notion of similarity to the satisfaction of variants of CTL (computation tree logic) formulas in a similar way as we do with $\mathcal{L}\natural$. That is, they define a notion of δ-weakening of timed CTL formulas. Here, like in our approach they weaken the properties w.r.t. to timing behavior on the until operators. However, as they are only considering timed systems, no transformations to the state formulas are performed.

Alfaro et al. [dAFS09] study linear and branching time distances on metric transition systems. For the linear distance they show that quantitative LTL formulas are preserved and they propose a variant of LTL that can be used for a logical characterization of this notion of trace distance. Similar results for branching time distances and quantitative CTL are presented.

In contrast to our approach the metric transition systems do not feature any real-time behavior. Therefore, the notions studied by Alfaro et al. do not consider deviations in timing behavior. Consequently, quantitative LTL is different from our logic $\mathcal{L}_{\natural}$ as they do not consider annotated temporal operators.

Fainekos and Pappas [FP09] and Donzé and Maler [DM10] studied robust satisfaction of linear-time metric temporal logic formulas. That is Fainekos and Pappas allow for bounded spatial deviations and Donzé and Maler allow, like in our work for deviations in space and time. Both compute, given a trajectory and a formula a "degree" of satisfaction of the formula. That way trajectories that are close w.r.t. to their measure still satisfy the formula, but not necessarily as robustly as before. Thus, their goal is to rate single trajectories of a running (or simulated) system in the sense that they compute a distance to a set of requirements. In contrast to that, our approach is meant to give a refinement relation on systems, such that we can construct properties and formulas syntactically that are preserved by all robust refinements.

Banach et al. [BZSH12] study so called retrenchments. Retrenchments weaken the guarantees of refinement in the sense that they allow for some deviations in the system outputs. However, in contrast to our approach, in the continuous world they allow for some arbitrary deviations w.r.t. variable valuations for a limited amount of time. Furthermore, they do not allow for temporal deviations in the system behaviors. Overall, their retrenchment gives less guarantees w.r.t. what properties that are preserved than refinement and robust refinement.

Note that abstractions also induce a notion of similarity. That is, all systems that have the same abstraction are considered similar. Consider for example the work of Moor et al. [MRO02] on discrete abstractions of hybrid systems. They consider an abstraction of the output trajectories of systems that partitions the state space into finitely many distinguishable valuations. The behavior of the system is then viewed as the sequence of regions the system passes through while it is evolving. This approach induces a similarity notion where all systems are considered similar that produce traces that feature the same order of events, i.e., regions visited. In contrast to their work we build our "abstractions" along the continuous trajectories instead of applying a fixed grid to discretize the observable behavior. The advantage is that we can provide a syntactic transformation on real-time temporal logic that characterizes which properties are preserved. Further, we retain more information about the temporal order of events

within the static regions used by Moor et al.. On the other hand, using our notion of similarity if the specification has trajectories that move closely to the border of state set that should be avoided, we have to choose the bound on the spatial deviation δ sufficiently small in order to be sure that trajectories of similar systems do not reach into that state set. With a pre-defined grid we could create a sharp line in the approximation in order to distinguish trajectories that reach the bad states and those which do not reach the bad states. However, our approach seems to be more natural in the sense that we want specifications to be robust w.r.t. small perturbations. Any hard borders would make us "blind" against whether we are close to the border of the region or far away. Thus, the grid would eventually accumulate more small regions around the borders of the bad states anyway in order to get stronger statements about how closely trajectories get to the bad states. Overall, the goals of the two approaches are different. Moor et al. want to synthesize a controller such that a given system is well-behaved whereas our goal is to show that some system is well-behaved as it is a robust refinement of a well-behaved specification.

Stauner [Sta02] defines a refinement relation on hybrid systems that we call 0-δ-refinement in this thesis. We refer to Example 5 (see page 37) for an example of two systems that are not similar w.r.t. this notion. Stauner further presents an approach based on specifications where guards and evolution domain constraints are overlapping in a single point. Starting from such a specification, he constructs a relaxed model where there is some slackness with respect to the variable valuations while switching modes. He then establishes a refinement between this relaxed model and an implementation that only changes its variables discretely according to numerical approximation of the original differential equations. The relaxed version of the model is basically an overapproximation of the possible behaviors on a syntactic level. Stauner's approach has the advantage that he can synthesize implementations syntactically. However, this comes at the price that there are certain restrictions necessary w.r.t. the specifications that can be refined. Our approach, does not need such artificial restrictions, and is able to deal with nondeterminism w.r.t. the actual switching points. Instead of synthesizing discrete time implementations, we will cover the topic of showing similarity for two given systems in Chapter 5.

3.4 Conclusion

In this chapter we presented a family of quantitative notions of similarity and robust refinement that allow us to consider systems as similar that differ in timing behavior as well as variable values. Although the idea to cover both aspects is not new [Dav09, DM10], we presented a slightly different version of this notion than what can be found in the literature. We decided to keep the bounds of the temporal deviation and of the spatial deviation separate in order to later use these parameters to modify the properties preserved. This led to an intuitive syntactic transformation on a rich class of logical properties. Further, we were able to identify cases, where no upper bound on the temporal deviation is necessary in order to transfer properties of interest. Overall, the notion presented in this chapter seems worthwhile as it is able to transfer a large number of properties formulated in our variant of metric temporal logic $\mathcal{L}_{\natural}$. Still it is general in the sense that other notions can be seen as special cases of this framework. Observe that in the case where $\varepsilon = 0$ and $\delta = 0$ our robust refinement relation coincides with classical refinement. Thus, we designed the transformation functions a to be the identity transformation for that case in order to yield the strongest possible results.

CHAPTER FOUR

Differential Dynamic Game Logic

By playing games you can artificially speed up your learning curve to develop the right kind of thought processes.

— Nate Silver

Contents

An important question when analyzing complex physical systems is whether one component is able to meet a given safety requirement. However, in many cases the component is not able to control the whole system but additional influences from the environment occur. Therefore, we want to answer this question in a way that we are sure that the component operates safely no matter what its environment does. Consider an autonomous robot moving around in a robotic factory environment. Global decision planning is infeasible, so the robot has limited knowledge about what the other elements of the factory will decide to do. If there is any probabilistic information about the decisions of agents, stochastic system models can be used for verification [Pla11]. Otherwise, the question can be considered as a game between the component and its environment. This game theoretic extension of hybrid systems is called hybrid games.

Hybrid games [QFD11,TLS00,HHM99,BBC09,VPVD11] have two types of actions: discrete jumps, which update the value of a variable instantaneously, and continuous evolutions along solutions of differential equations. Time only passes for the latter action. Hence hybrid games are a natural extension of timed games [MPS95] which only support clocks with differential equation $\dot{x} = 1$ and only allow variables to be reset to 0 and not assigned arbitrarily.

Using differential dynamic logic dℒ, we can express simple games. For example, when F is a hybrid system describing a factory and R a hybrid system describing a robot, then a formula of the form $[F]\langle R\rangle \mathit{safe}$ can be used to express that, for all behaviors of a factory F, the robot R can choose at least one behavior ensuring safety (represented by some dℒ formula *safe*). This is a simple game expressible in dℒ, but it stops after one round of interactions by the factory player and the robot player. In order to say that the robot is still safe if it reacts appropriately after the factory changed its mind in response to the robot's first choice, we can use the formula $[F]\langle R\rangle(\mathit{safe} \wedge [F]\langle R\rangle \mathit{safe})$. We can do so for any given number of rounds of interactions of F and R, but we typically want to say that the system will be safe for *any* number of interactions of F and R, not just for two.

Contributions. In this chapter, we propose an extension to dℒ, differential dynamic game logic (dDGℒ), that can state those properties using several game constructs, including repetition operators $(G)^{[*]}$ and $(G)^{\langle * \rangle}$ to say that game G repeats. The difference between both operators is which player

decides how often to repeat the game. They decide how often to repeat before the game starts. For example, the dDGL formula $([F]\langle R\rangle)^{[*]}safe$ expresses that, no matter how often the player responsible for $(\cdot)^{[*]}$ decides to repeat the game $[F]\langle R\rangle$, the state resulting from those alternating choices by F and R is safe.

In order to prove such properties, we lift the induction principles of dL to dDGL. A dDGL formula $(G)^{[*]}\phi$ behaves in some ways like the dL or dDGL formula $[\alpha^*]\phi$ (where α^* is the hybrid system that repeats α). In both cases, we consider all possible numbers of iterations, because we do not know how often it will be repeated. The dDGL formula $(G)^{\langle*\rangle}\phi$ has similarities to the dL formula $\langle\alpha^*\rangle\phi$, since in both cases, we can choose some number of repetitions. Yet, G is a hybrid game, whereas α is a hybrid system. Nevertheless, we show that the induction principles of invariants and variants lift from dL to dDGL.

We prove that dDGL is a conservative extension of dL and, thus, our theorem prover KeYmaera [PQ08a] for dL can be extended such that it can be used to prove hybrid games expressible in dDGL. We develop a proof calculus for the specifics of dDGL and implement it in KeYmaera. We develop and verify a case study in which a mobile robot satisfies a joint safety and liveness objective in a factory automation scenario, in which the factory may perform interfering actions.

Structure of the Chapter. This chapter follows the presentation from our previous work published in [QP12a, QP12b]. We first define the syntax of dDGL in Section 4.1 and semantics in Section 4.2. Then, we present a sequent proof calculus for dDGL in Section 4.3. In Section 4.4 we show the results of our factory automation case study. Related approaches are discussed in Section 4.5 and we conclude this chapter in Section 4.6.

4.1 Syntax

Based on hybrid programs we define *hybrid games*. The idea behind our notion of hybrid games is to use operators similar to those of hybrid programs, but for games on top of hybrid systems. The particular hybrid games that we consider here are two-player games produced by the following *grammar* (α is a hybrid program):

$$G ::= [\alpha] \mid \langle\alpha\rangle \mid (G_1 \cap G_2) \mid (G_1 \cup G_2) \mid (G_1 G_2) \mid (G)^{[*]} \mid (G)^{\langle*\rangle}$$

By $\mathcal{G}$, we denote the *set of all such hybrid games*. The intuition behind these games is as follows. The game is played by two players, which we call *Verifier* and *Falsifier* who play by the following rules: In the game $[\alpha]$ Falsifier resolves the nondeterminism within α whereas in the game $\langle\alpha\rangle$ Verifier is allowed to do so. Observe that our notion of hybrid games is built *on top of hybrid systems*, that is, every hybrid program α is, by way of $[\alpha]$ or $\langle\alpha\rangle$, directly a hybrid game. The game $(G_1 G_2)$ is the sequential composition of games, where game G_2 is played right after game G_1 has finished. In a game $(G_1 \cap G_2)$ Falsifier may decide whether the game proceeds with G_1 or with G_2. In the game $(G_1 \cup G_2)$ this choice is made by Verifier. Repetitive game playing is possible using the iteration constructs $(G)^{[*]}$ and $(G)^{\langle*\rangle}$, where, for the first one, Falsifier decides how many iterations are played and, for the latter one, Verifier makes the choice. We request that the choice on the number of iterations has to be made by *advance notice*. That is, the player responsible for controlling the iteration decides how often G is repeated when the game starts and announces it to the other player.

Winning conditions for the games are formulated in dDG$\mathcal{L}$ as postconditions of games. A *strategy* for a player determines how to resolve the nondeterminism under his control based on the state reached by the game played so far. The nondeterminisms inside a hybrid system are resolved by Falsifier or Verifier depending whose turn it is by choosing which real values to assign to x when executing $x := *$ statements, which branch to follow for choices $\cup$, which number of loop iterations to choose, and how long to follow continuous flows.

The dDG$\mathcal{L}$*-formulas* are first-order formulas over the reals extended by hybrid games. They are defined by the following grammar (θ_i are terms, x is a variable, $\sim \in \{<, \leq, =, \geq, >, \neq\}$, ϕ and ψ are formulas, and G is a hybrid game):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid G\, \phi$$

A dDG$\mathcal{L}$ formula $G\,\phi$ is valid if Verifier has a strategy to ensure that ϕ holds after playing the game G. Therefore, the goal of Verifier is to make ϕ true while that of Falsifier is complementary, i.e., to make ϕ false. Note that the formula ϕ itself might contain another game.

Consider the dDG$\mathcal{L}$ formula $([\alpha])^{\langle*\rangle}\phi$, which expresses that Verifier can choose a number of repetitions n, such that the formula ϕ holds after these n repetitions of α in which Falsifier resolves the nondeterminism. Note that

this dDGL formula is not equivalent to $[\alpha^*]\,\phi$, which would demand that it holds for all possible numbers of executions of α. It is also not equivalent to $\langle\alpha^*\rangle\,\phi$ as this would give control to Verifier over the (possibly unbounded) nondeterminism during the executions of α. A similar observation can be made for $(\langle\alpha\rangle)^{[*]}\phi$, which that for any number of iterations chosen by Falsifier, Verifier is always able to ensure that after this number of iterations of α the formula ϕ holds by making appropriate choices of the nondeterminisms in α. Combining the repetition operator and the choice operators we can express properties like $(\langle\beta\rangle\,[\alpha] \cup [\alpha]\,\langle\beta\rangle)^{[*]}\phi$. This formula means that ϕ holds after any number of iterations (as Falsifier has control over the number of iterations) while Verifier can control (by $\cup$) for each iteration if he wants to move first according to β or Falsifier has to move first according to α.

4.2 Semantics

Next, we define the *semantics of* dDGL. Actually, we define two different semantics. A classical modal semantics in Section 4.2.1 that does consider games just as sequences of modalities and a game semantics in Section 4.2.2 that formalizes notions of strategy, play, and winning. In Section 4.2.3 we show that both these semantics are equivalent in the sense that if for a state ν a formula is satisfied in the modal semantics then Verifier has a strategy to win the game from this state no matter what Falsifier does.

4.2.1 Classical Modal Semantics

Recall from Chapter 2 that for a set of variables V, we denote by $Sta(V)$ the set of states, i.e., all mappings of type $V \to \mathbb{R}$. Let $\nu(\theta)$ denote the valuation of a term θ in a state ν.

Definition 34 (Semantics of dDGL formulas)**.** *The* semantics $\models$ *of a* dDGL *formula w.r.t. state ν uses the standard meaning of first-order logic:*

1. $\nu \models \theta_1 \sim \theta_2$ *iff* $\nu(\theta_1) \sim \nu(\theta_2)$ *for* $\sim \in \{<, \leq, =, \geq, >\}$
2. $\nu \models \phi \wedge \psi$ *iff* $\nu \models \phi$ *and* $\nu \models \psi$*, accordingly for* $\neg, \vee, \rightarrow, \leftrightarrow$
3. $\nu \models \forall x\,\phi$ *iff* $\omega \models \phi$ *for all* ω *that agree with* ν *except for the value of* x

4. $\nu \models \exists x\, \phi$ *iff* $\omega \models \phi$ *for some* ω *that agrees with* ν *except for the value of* x

Statements about hybrid games G *and programs* α *have the following semantics*

5. $\nu \models [\alpha]\phi$ *iff* $\omega \models \phi$ *for all* ω *with* $(\nu, \omega) \in \rho(\alpha)$,
6. $\nu \models \langle\alpha\rangle\phi$ *iff* $\omega \models \phi$ *for some* ω *with* $(\nu, \omega) \in \rho(\alpha)$,
7. $\nu \models (G_1 \cup G_2)\phi$ *iff* $\nu \models G_1\phi$ *or* $\nu \models G_2\phi$,
8. $\nu \models (G_1 \cap G_2)\phi$ *iff* $\nu \models G_1\phi$ *and* $\nu \models G_2\phi$,
9. $\nu \models (G_1 G_2)\phi$ *iff* $\nu \models G_1(G_2\phi)$,
10. $\nu \models (G)^{[*]}\phi$ *iff* $\nu \models (G^n)\phi$ *holds for all* $n \in \mathbb{N}$,
11. $\nu \models (G)^{\langle*\rangle}\phi$ *iff* $\nu \models (G^n)\phi$ *holds for some* $n \in \mathbb{N}$,

where G^n *denotes the* n*-times sequential composition of* G *and* $G^0\phi \equiv \phi$. *A formula* ϕ *is valid (denoted by* $\models \phi$*) iff* $\nu \models \phi$ *for all states* $\nu \in Sta(V)$.

Observe that the semantics for quantifiers and the box and diamond modalities is identical to that of dℒ. For choices of Verifier it is sufficient that one of the choices satisfies the postcondition whereas for choices of Falsifier it is important that both do so. Sequential composition can be handled inductively by moving the second part into the postcondition. For the repetition operators we have that if Verifier is in control of the number of loops, there has to be at least one unwinding such that the postcondition holds. On the other hand, for Falsifier this has to be the case for every possible unwinding.

4.2.2 Game Semantics

The previous definitions are abstract in the sense that they do not refer to how games are actually played. Therefore, we now provide a structural operational semantics for games, formally define the notions of play and strategy, and then prove that the existence of a winning strategy for Verifier coincides with the notion of validity in Definition 34. This is, we provide a *game semantics* for dDGℒ.

Definition 35 (Game Position)**.** *For a game G and a state ν, we use $G@\nu$ to denote that the game is in the* position *where starting from state ν the game will follow the transitions of G. We denote by $\mathcal{P}_{\mathcal{G}} := \mathcal{G} \times Sta(V)$ the* set of all game positions*.*

We add special games $\surd$, $\top$, and $\bot$ to denote the possible outcomes of a game. In the latter two cases either of the players was unable to make another move. That is, we denote by $\top$ the case where Falsifier failed to make a move and by $\bot$ the case where this happened to Verifier. This can happen in cases where the program specifying the current action contains a test that is not satisfiable. For example, the game $[?false]$ will terminate in $\top$ to indicate that it is always won by Verifier whereas the game $\langle ?false\rangle$ will terminate in $\bot$ to indicate that it is always won by Falsifier. Otherwise, the game terminates in $\surd$ after the players played all their actions.

Definition 36 (Extended Game Position)**.** *We denote the* set of all extended game positions *by*

$$\mathcal{P}_{\mathcal{E}} := (\mathcal{G} \cup \{\surd, \top, \bot\}) \times Sta(V) \ .$$

First, as we want to base our semantics on hybrid traces, we define the set of all traces possible for a program starting in the state ν.

Definition 37 (Initialized Traces)**.** *For a hybrid program α the* set of all traces *that start in the initial state ν is given by*

$$\tau(\alpha, \nu) = \{\sigma \mid \sigma(0) = \nu \wedge \sigma(max(\text{dom } \sigma)) \neq \Lambda \wedge \sigma \in \tau(\alpha)\} \ .$$

Recall from Definition 1 (see page 17) that Λ denotes the failure state reached in case of failed tests. Furthermore, we denote the empty trace by $()$.

The operational semantics for the games is structured into three types of actions: those controllable by Falsifier (prefixed with F), those controllable by Verifier (prefixed with V), and those for modeling sequential composition (prefixed with S).

Definition 38 (Structural Operational Semantics of Games)**.** *For a game G its* operational semantics $[\![G]\!]$ *is given by the rules defined in Figure 4.1. The semantics provides a relation between game positions and traces, i.e.,*

$$[\![G]\!] \subseteq \mathcal{P}_{\mathcal{G}} \times \mathfrak{T} \times \mathcal{P}_{\mathcal{E}} \ .$$

Furthermore, we define two subsets of this semantics. We denote by $[\![G]\!]_F$ *the relation between all states defined only by the* F *and* S *rules. By* $[\![G]\!]_V$ *we denote the relation defined by the* V *and* S *rules.*

Note that in Figure 4.1, G^n refers to following definition.

Definition 39. *The* n*-times sequential composition of a game* G *denoted by* G^n *is defined as follows:*

$$G^0 \hat{=} \surd$$
$$G^1 \hat{=} G$$
$$G^n \hat{=} (G\ G^{n-1})$$

The rules of the structural operational semantics define a transition system that is possibly uncountably branching, due to the nondeterminism in hybrid programs, for instance, in choosing evolution times. This is accounted for in rule F1. Rule F1 says that if there is a trace σ of α starting in initial state ν then Falsifier can make a step from position $[\alpha]@\nu$ playing trace σ ending in the special position $\surd@last(\sigma)$. Here $\surd$ is the game that denotes termination. Rule F2, on the other hand, covers the case where there is no run of α at all that originates in ν. In those cases Falsifier automatically looses, i.e., his only move is towards the state $\top$. If there is a choice between games G and H to be made by Falsifier, he can use rules V3 or V4 to choose either G or H. For repetitions, Falsifier chooses a number n of iterations and uses rule F5 to proceed to position $G^n@\nu$ where the position indicates that the game G has been unwinded n-times.

Verifier uses the V instead of the F rules. Still, those work the same but on the operators under his control. For sequential composition we further provide the S rules. The rules S2-S4 cover the terminating cases. In the case where the first component of the sequential composition terminates the game moves to a position where the second component is played using rules S2. If one of the players fail to make a move then the second component is skipped and the game stays in the position indicating that the other player has won. That is, if Falsifier is not able to make a move then the game will eventually end in $\top$and if Verifier is not able to make a move then it will end in $\bot$. In case the first component of the sequential composition has a more complex structure the rule S1 can be used. For instance, if $G = (G_1\ G_2)$ is itself a sequential composition then $(G\ H)$ might progress

$$\text{(F1)}\ \frac{\sigma \in \tau(\alpha,\nu)}{[\alpha]@\nu \xrightarrow{\sigma} \surd@last(\sigma)} \qquad \text{(F2)}\ \frac{\tau(\alpha,\nu)=\emptyset}{[\alpha]@\nu \xrightarrow{()} \top@\nu}$$

$$\text{(F3)}\ \frac{}{G \cap H@\nu \xrightarrow{()} G@\nu}$$

$$\text{(F4)}\ \frac{}{G \cap H@\nu \xrightarrow{()} H@\nu} \qquad \text{(F5)}\ \frac{n \in \mathbb{N}}{(G)^{[*]}@\nu \xrightarrow{()} G^n@\nu}$$

$$\text{(V1)}\ \frac{\sigma \in \tau(\alpha,\nu)}{\langle\alpha\rangle@\nu \xrightarrow{\sigma} \surd@last(\sigma)} \qquad \text{(V2)}\ \frac{\tau(\alpha,\nu)=\emptyset}{\langle\alpha\rangle@\nu \xrightarrow{()} \bot@\nu}$$

$$\text{(V3)}\ \frac{}{G \cup H@\nu \xrightarrow{()} G@\nu}$$

$$\text{(V4)}\ \frac{}{G \cup H@\nu \xrightarrow{()} H@\nu} \qquad \text{(V5)}\ \frac{n \in \mathbb{N}}{(G)^{\langle*\rangle}@\nu \xrightarrow{()} G^n@\nu}$$

$$\text{(S1)}\ \frac{G@\nu \xrightarrow{\sigma} G'@\omega}{(G\ H)@\nu \xrightarrow{\sigma} (G'\ H)@\omega} \qquad \text{(S2)}\ \frac{G@\nu \xrightarrow{\sigma} \surd@\omega}{(G\ H)@\nu \xrightarrow{\sigma} H@\omega}$$

where $G' \notin \{\surd, \bot, \top\}$

$$\text{(S3)}\ \frac{G@\nu \xrightarrow{\sigma} \bot@\nu}{(G\ H)@\nu \xrightarrow{\sigma} \bot@\nu} \qquad \text{(S4)}\ \frac{G@\nu \xrightarrow{\sigma} \top@\nu}{(G\ H)@\nu \xrightarrow{\sigma} \top@\nu}$$

Figure 4.1: Structural operational semantics of hybrid games (Verifier can only control V and S rules and Falsifier can only control F and S rules).

to a position $(G_2\ H)$ via some trace produced by G_1. That way, rule S1 ensures associativity of the sequential composition operator.

Observe that each path is of finite length, because the number of iterations is chosen nondeterministically but a priori. Note that, this semantics does not yet define who decides which options to follow. In particular, the structural operational semantics of $\cap$ and $\cup$ is still the same and that of $(\cdot)^{[*]}$ and $(\cdot)^{\langle*\rangle}$ is still the same, but they will differ as soon as we define which player gets to choose. The Verifier can choose V rules and the Falsifier can choose F rules. The S rules are determined anyway.

Example 11 (Branching)**.** *In Figure 4.2 the operational semantics for a simple game is sketched. Dashed lines illustrate choices to be determined by Verifier, straight lines are used in cases where the nondeterminism is resolved by Falsifier. Note that the reachability of the states $\surd$, $\top$, $\bot$ depends on the actual game and the current state.*

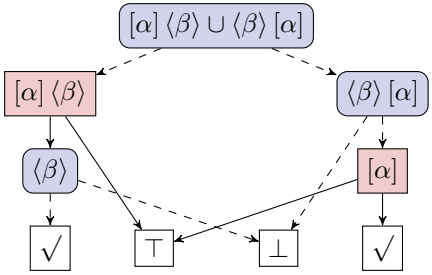


Figure 4.2: Explicit branching

In Figure 4.3 the same game is sketched adding the state component to the picture. Note that uncountably many structures of the same form exist in the semantics depending on the initial state that might or might not reach states depicted in the image. Let us consider how this connects to our structural operational semantics. When the game is in a position described by $([\alpha]\langle\beta\rangle \cup \langle\beta\rangle[\alpha])@\nu$, by rule V3 Verifier can make a move to position $[\alpha]\langle\beta\rangle@\nu$. Alternatively, Verifier could use rule V4 to move to a position $\langle\beta\rangle[\alpha]@\nu$. In the first case it is Falsifier's turn. If we assume that there is a transition $[\alpha]@\nu \xrightarrow{\sigma} \surd@\nu'$ then Falsifier can use the rules F1 and S2 to move to a position $\langle\beta\rangle@\nu'$. This can be done for each ν' creating possibly uncountably infinite branching depending on the structure of α.

In a position $\langle\beta\rangle[\alpha]@\nu$ Verifier is to determine the followup position. If we assume $\langle\beta\rangle@\nu \xrightarrow{\sigma} \surd@\nu'$ then Verifier can use the rules V1 and S2 to move to a position $[\alpha]@\nu'$. Again, this can cause uncountably infinite branching in this case depending on the structure of β.

Example 12 (Repetition with advance notice semantics)**.** *In Figure 4.4 the operational semantics for the repetition operator is shown. Again, dashed lines illustrate choices to be determined by Verifier, straight lines are used in cases where the nondeterminism is resolved by Falsifier. Here, all the dashed lines are possible by rule V5. For the straight lines we assume*

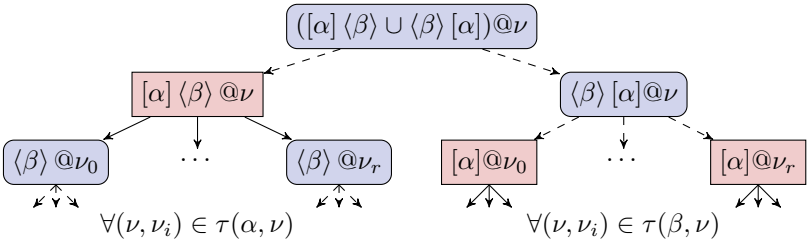


Figure 4.3: Explicit branching with states

that rule F1 is applicable and apply it together with S2 in order to get a terminating execution. Observe that, the loop causes countably infinite branching. However, every path only has finite depth.

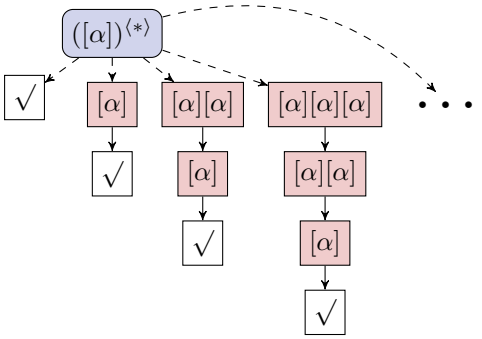


Figure 4.4: Repetition with advance notice semantics (only successfully terminating runs are depicted here)

Thus far we have formalized the possible moves. In order to formally capture the choices made by the players, we introduce the notion of strategy.

Definition 40 (Strategy)**.** *A* strategy $s : \mathcal{P}_{\mathcal{G}} \xrightarrow{\text{part}} \mathcal{P}_{\mathcal{E}} \times \mathfrak{T}$ *is a partial mapping between game positions and traces. A strategy s is called* compatible *with a game G if its actions are allowed, i.e., if $s(g@\nu) = (g', \omega, \sigma)$ then*

$$((g@\nu) \xrightarrow{\sigma} g'@\omega) \in [\![G]\!] \ .$$

Observe that either $\nu = \omega \wedge \sigma = ()$ or $\omega = last(\sigma)$. This follows directly from Definition 38. We now define how strategies specifically for Falsifier and Verifier look like.

Definition 41 (Specific strategies)**.** *We call* $f : \mathcal{P}_\mathcal{G} \xrightarrow{\text{part}} \mathcal{P}_\mathcal{E} \times \mathfrak{T}$ *a* Falsifier strategy *for the game* G *iff the following conditions are satisfied:*

1. *If* $f(g@\nu) = (g'@\omega, \sigma)$ *then* $((g@\nu) \xrightarrow{\sigma} g'@\omega) \in [\![G]\!]_F$.
2. *If* $((g@\nu) \xrightarrow{\sigma} h@\omega) \in [\![G]\!]_F$ *then there are* h', ω', σ' *such that*

$$f(g@\nu) = (h'@\omega', \sigma') \ .$$

We call $v : \mathcal{P}_\mathcal{G} \xrightarrow{\text{part}} \mathcal{P}_\mathcal{E} \times \mathfrak{T}$ *a* Verifier strategy *for the game* G *iff the following conditions are satisfied:*

1. *If* $v(g@\nu) = (g'@\omega, \sigma)$ *then* $((g@\nu) \xrightarrow{\sigma} g'@\omega) \in [\![G]\!]_V$.
2. *If* $((g@\nu) \xrightarrow{\sigma} h@\omega) \in [\![G]\!]_V$ *then there are* h', ω', σ' *such that*

$$v(g@\nu) = (h'@\omega', \sigma') \ .$$

The first condition ensures that the strategy is compatible w.r.t. the transition system described by the F (resp. V) and S rules. That is, it only chooses follow-up positions that are reachable in that transition system. The second condition ensures that the strategy is total in the sense that whenever there is a follow-up position defined by F (resp. V) and S rules then it chooses one.

Example 13. *Consider the following example of a hybrid program* α*:*

$$\alpha \hat{=} (x := 1 \cup x := 0)$$

The traces of α *are given by*

$$\tau(\alpha) = \bigcup_{\nu \in Sta(V)} (\{\hat{\nu} \circ (\{(0, \nu[x \mapsto 1])\})\} \cup \{\hat{\nu} \circ (\{(0, \nu[x \mapsto 0])\})\})$$

Therefore, the possible strategies for Falsifier in the game $[\alpha]$ *are limited by this choice of traces. For a position* $[\alpha]\,@\nu$ *the choice consists of exactly two possible traces. A possible Falsifier strategy can now be defined as follows:*

$$f([\alpha]\,@\nu) = (\surd @\nu[x \mapsto 0], \hat{\nu} \circ (\{(0, \nu[x \mapsto 0])\}))\ \ f.a.\ \nu \in Sta(V)$$

Using this notion of strategy, we now formalize the rules of the game by determining which player gets to choose from the actions of the operational semantics.

Definition 42 (Play)**.** *Assume that the empty trace* () *is the neutral element w.r.t. trace composition. Given a Falsifier strategy* f*, and a Verifier strategy* v*, a* play $\mathfrak{p}_{f,v}(\cdot) : \mathcal{P}_{\mathcal{G}} \to (\mathcal{G} \cup \{\surd, \top, \bot\}) \times \mathfrak{T}$ *is defined by applying* $s := f \cup v$ *exhaustively. That is*

$$\mathfrak{p}_{f,v}(G@\nu) = \begin{cases} (G, ()) & \text{if } s(G@\nu) = \text{undef} \\ \big((\surd, s|_3(G@\nu)) & \\ \circ(\mathfrak{p}_{f,v}(s|_1(G@\nu)@s|_2(G@\nu)))\big) & \text{otherwise} \end{cases}$$

where $s|_i$ *denotes the projection of the result of* s *to its* i*-th component, and the composition* $\circ : \mathcal{G} \times \mathfrak{T} \times \mathcal{G} \times \mathfrak{T}$ *is defined by*

$$(G, \sigma) \circ (G', \sigma') := (G', \sigma \circ \sigma')\ .$$

Here, $s|_1$ denotes the current game, $s|_2$ denotes the state of the current position, and $s|_3$ denotes the trace component.

Observe that in case the play successfully terminates then it produces a non-empty trace. Further, note that for handling sequential composition it is crucial to pass the continuation to the strategies in order to allow for different choices depending on the future game positions. This also enables the strategies to react on the chosen number of loop iterations.

Example 14. *Continuing Example 11, in Figure 4.5 the effect of the strategy choices in combination with our notion of play is sketched. For each of the positions, the strategy determines the follow-up position. Here the arrows marked by* f *denote the directions chosen by Falsifier and those marked by* v *denote those chosen by Verifier.*

Now that we have a mathematical notion of a play, we define who is the winner of such a play.

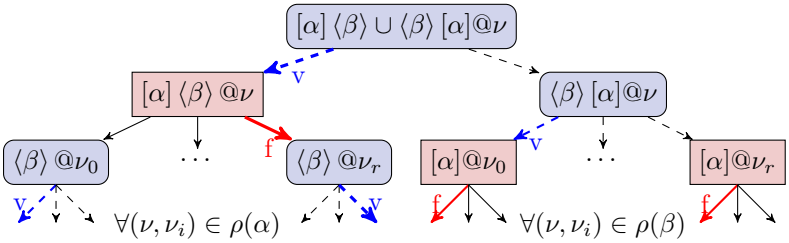


Figure 4.5: Example for the effect of the strategies

Definition 43 (Winning)**.** *Consider a game G and a* dDGℒ *formula ϕ as winning condition. For an initial state ν, two strategies v and f, the game G is* won *by Verifier iff G ends in a position $\mathfrak{p}_{f,v}(G@\nu) = (H, \sigma)$ where either $H = \surd$ and $last(\sigma) \models \phi$, or $H = \top$. Otherwise, Falsifier wins (i.e., the game is zero-sum). For a* dDGℒ*-formula ϕ a strategy s is called* winning *for a game G if, by applying this strategy, Falsifier (resp. Verifier) wins every play of G regardless of which strategy Verifier (resp. Falsifier) follows.*

Now the choice of the symbols $\top$ and $\bot$ should become apparent. In the case where the game ends in the state $\top$ the Falsifier was unable to make a move. This equivalent with a test failing in a subformula of the form $[?F]\phi$. Thus, this subformula is trivially satisfied. In the case where we end in a state $\bot$ the same happened to Verifier, that is he failed to make a move due to the fact in a subformula of the form $\langle?F\rangle\phi$ the test condition could not be satisfied.

Note that we just oversimplified the fact a bit. It could also be the case that the failing test was written in form of an evolution domain constraint in a differential equation system. Still, the intuition behind these cases remains the same.

Example 15. *Continuing Example 13, we can now see that the strategy provided is a winning strategy if the winning condition is for instance $x = 0$.*

4.2.3 Semantics Relation

As promised, we now link the semantics of formulas given in Definition 34 with the operational semantics of games. This is an important result as we

will prove the soundness of our proof calculus w.r.t. to the classical modal semantics but want to be sure that this semantics matches our intuition behind how the games should be played.

Theorem 4. *For a state ν, $\nu \models G\phi$ iff Verifier has a strategy in the game G with winning condition ϕ starting in position $G@\nu$ such that he wins the game.*

Proof. Our proof is by structural induction.

- Base case:
  1. $G \equiv [\alpha]$: If $\tau(\alpha, \nu) = \emptyset$ then the game ends in $\top$, by rule F2 and Verifier wins. Otherwise, if there is a strategy v that wins the game from position $G@\nu$ and ends in position $\surd@\omega$, due to rule F1 $\sigma \in \tau(\alpha, \nu)$ with $last(\sigma) = \omega$. By Definition 43 we know that $\omega \models \phi$. Therefore, by Definition 34, $\nu \models [\alpha]\phi$.

     Assume $\nu \models G\phi$. From Definition 34 we know that, since we have $\nu \models [\alpha]\phi$, $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$. By definition of $\rho(\alpha)$ we know, that there is no trace $\sigma \in \tau(\alpha, \nu)$ with $last(\sigma_\omega) = \omega'$ and $(\nu, \omega') \notin \rho(\alpha)$. Therefore every compatible strategy for Verifier wins.
  2. $G \equiv \langle\alpha\rangle$: If there is a strategy v that wins the game from position $G@\nu$ and ends in position $\surd@\omega$, then by Definition 43 $\omega \models \phi$, and due to rule F1 $\sigma \in \tau(\alpha, \nu)$, and thus $(\nu, \omega) \in \rho(\alpha)$. Therefore by Definition 34 $\nu \models \langle\alpha\rangle\phi$.

     Assume $\nu \models G\phi$. From Definition 34 we know that, since we have $\nu \models \langle\alpha\rangle\phi$, $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$. Therefore, let the strategy of Verifier be defined by

     $$v(\langle\alpha\rangle@\nu) = (\surd@\omega, \sigma)$$

     for $\sigma \in \tau(\alpha, \nu)$ with $last(\sigma) = \omega$. This strategy is compatible by rule V1.

- IH: Let the property be satisfied for all structurally simpler games.
- Induction Step

1. $G \equiv (G_1 \cup G_2)$: Assume that $\nu \models (G_1 \cup G_2)\phi$. Consider the case where $\nu \models G_1\phi$, then we set $v((G_1 \cup G_2)@\nu) = (G_1@\nu, ())$, which is compatible according to rule V3. By Definition 34, we otherwise know $\nu \models G_2\phi$ and set $v((G_1 \cup G_2)@\nu) = (G_2@\nu, ())$, which is compatible by rule V4. By IH, these strategies can be extended, thus that they win the game for Verifier.

   Conversely, assume there is a Verifier strategy v that wins the game $G_1 \cup G_2$ with winning condition ϕ for Verifier from $G@\nu$. Case distinction: If $v(G@\nu) = (G_1@\nu, ())$ then, by IH, $\nu \models G_1\phi$. Otherwise, if $v(G@\nu) = (G_2@\nu, ())$, then, by IH, $\nu \models G_2\phi$. These are the only choices that yield a compatible strategy (by Definition 38). Therefore, $\nu \models G_1\phi$ or $\nu \models G_2\phi$. Which is equivalent to $\nu \models (G_1 \cup G_2)\phi$ by Definition 34.

2. $G \equiv (G_1 \cap G_2)$: Assume that $\nu \models G_1 \cap G_2\phi$. In this case $\nu \models G_1\phi$ and $\nu \models G_2\phi$ by Definition 34. By IH, there is a strategy for Verifier for both cases.

   Conversely, assume the existence of a strategy for Verifier. In the game $(G_1 \cap G_2)$, this strategy has no influence on the first choice made, because Falsifier decides the first move. Therefore, both branches (defined by rules F3 and F4) have to be evaluated. This means that, by IH, $\nu \models G_1\phi$ and $\nu \models G_2\phi$, which is equivalent to $\nu \models (G_1 \cap G_2)\phi$ by Definition 34.

3. $G \equiv (G_1G_2)$: There are three possible cases how the game could progress defined by the rules S1, S2, S3, and S4. Assume that $\nu \models G\phi$. From Definition 34 we know that $\nu \models G_1(G_2\phi)$. Let $\psi \equiv G_2\phi$. By IH, there is a strategy v_1 such that Verifier wins $G_1@\nu$ with winning condition ψ. This also means, again by IH, if $\surd@\omega = \mathfrak{p}_{s_\Box, v_1}(G_1@\nu)$ there is a strategy $v_{2,\omega}$ such that for every compatible strategy $s_\Box$ of Falsifier,Verifier wins $G_2@\omega$ with winning condition ϕ. In particular, Verifier wins $G\phi$ by following strategy v_1 first and then following $v_{2,\omega}$ whenever the outcome of G_1 was ω. If $\top@\omega = \mathfrak{p}_{f,v_1}(G_1@\nu)$, Verifier wins immediately by rule S4 and we do not need to play G_2. It is never the case that $\bot@\omega = \mathfrak{p}_{f,v_1}(G_1@\nu)$ because that would contradict the assumption that v_1 wins the game G_1 with winning condition ψ from position $G_1@\nu$.

Conversely, assume that there is a strategy for Verifier such that for every strategy of Falsifier he wins the game G with winning condition ϕ if it is started in position $G@\nu$. This means either the strategy can force the game G_1 started in position $G_1@\nu$ to terminate in a position $\top@\omega$. If this is the case then, by IH, $G_1\psi$ is valid for every ψ. Therefore, it is valid for $\psi \equiv G_2\phi$. If the game G_1 ends in a position $\surd@\omega$, then we know that the strategy wins the game G_2 with winning condition ϕ started in a position $G_2@\omega$. By IH, this means that $\omega \models G_2\phi$. As the strategy led to the position from $G_1@\nu$ we know, by IH, that $\nu \models G_1(G_2\phi)$. This, by Definition 34, gives that $\nu \models (G_1G_2)\phi$.

4. $G \equiv (G_1)^{[*]}$: Assume $\nu \models (G_1)^{[*]}\phi$, then by Definition 34, we know that $\nu \models G_1^n$ for every $n \in \mathbb{N}$. By IH, this means that for every $n \in \mathbb{N}$ there is a strategy for Verifier s_n that wins the game started in position $G^n@\nu$.

   Construct a strategy for G the following way:

   For a game G its closure under subgame, $cl(G)$, is defined inductively as:

   - $cl([\alpha]) = \{[\alpha]\}$ and $cl(\langle\alpha\rangle) = \{\langle\alpha\rangle\}$
   - $cl(G_1G_2) = \{G_1G_2\} \cup cl(G_1) \cup cl(G_2)$
   - $cl(G_1 \cup G_2) = \{G_1 \cup G_2\} \cup cl(G_1) \cup cl(G_2)$
   - $cl(G_1 \cap G_2) = \{G_1 \cap G_2\} \cup cl(G_1) \cup cl(G_2)$
   - $cl((G)^{[*]}) = \{(G)^{[*]}\} \cup \bigcup_{n\in\mathbb{N}} cl(G^n)$
   - $cl((G)^{\langle*\rangle}) = \{(G)^{\langle*\rangle}\} \cup \bigcup_{n\in\mathbb{N}} cl(G^n)$

   Let $\tilde{s}_n$ denote $s_n \cap ((cl(G^n) \setminus cl(G^{n-1})) \times Sta(V) \times \mathcal{P}_{\mathcal{E}} \times \mathfrak{T}$ where

   $$cl(G^0) = cl(G^{-1}) = \emptyset \ .$$

   The subtraction restricts the domain of the strategy to the parts that are used to handle the first iteration of the loop, and ends in a state where the next game to play is G^{n-1}. At that position we apply its respective strategy. Therefore, the strategy defined by $s \mathrel{\hat{=}} \bigcup_{n\in\mathbb{N}} \tilde{s}_n$ wins the game for Verifier when started in position $G@\nu$.

   Conversely assume the existence of a strategy that wins the game started in position $G@\nu$ with winning condition ϕ for Verifier. For the operator $(\cdot)^{[*]}$, Falsifier decides how often to repeat

G, so we have to explore all possible numbers of repetitions produced by rule F5. This is, for all $n \in \mathbb{N}$, the strategy wins the game G^n started from position $G_1^n@\nu$. By IH, this means $\nu \models G_1^n\phi$ for all $n \in \mathbb{N}$. This implies $\nu \models (G_1)^{[*]}\phi$ by Definition 34.

5. $G \equiv (G_1)^{\langle * \rangle}$: Assume $\nu \models (G_1)^{\langle * \rangle}\phi$, then by Definition 34, we know that $\nu \models G_1^n$ for some $n \in \mathbb{N}$. Then we choose as strategy for Verifier $v((G_1)^{\langle * \rangle}@\nu) = (G_1^n@\nu, ())$ which by IH can be extended to a strategy that wins the game.

   Conversely, assume the existence of a strategy v that wins the game started in position $G@\nu$ with winning condition ϕ for Verifier. Let $v(G@\nu) = G_1^n@\nu$ for some $n \in \mathbb{N}$ (according to Definition 38 and V5, these are the only compatible moves). By IH, this implies that $\nu \models G_1^n\phi$ for that $n \in \mathbb{N}$, which, further implies $\nu \models (G_1)^{\langle * \rangle}\phi$ by Definition 34. □

Corollary 6. *The formula $G\phi$ is valid iff Verifier has a winning strategy in the game G for the winning condition ϕ.*

A crucial point for the design of dDGL is that we want it to be conservative with respect to differential dynamic logic in the sense that all dL formulas are dDGL formulas and that any dL formula is valid in the semantics of dDGL if and only if it was valid in the original semantics for dL. This allows us to transfer soundness results for proof calculus rules from dL to dDGL and extend our theorem prover KeYmaera with additional proof rules for handling the extra dDGL constructs in addition to dL operators.

Definition 44. *A logic A is a* conservative extension *of logic B if all formulas of B are formulas of A and valid w.r.t. the semantics of A if, and only if, they are valid w.r.t. the semantics of B.*

Theorem 5 (Conservative Extension)**.** *Differential dynamic game logic is a conservative extension of differential dynamic logic.*

Proof. The semantics for dDGL-formulas that only contain games of the form $[\alpha]$ and $\langle\alpha\rangle$, which are exactly the modalities originally occurring in dL, coincides with the dL semantics of these formulas. □

4.3 Proof Rules for dDG$\mathcal{L}$

In this section, we present a sound but incomplete sequent proof calculus for dDG$\mathcal{L}$. It is incomplete, because reachability in hybrid systems is not even semidecidable [Pla10b]. The calculus symbolically executes the hybrid games and hybrid programs. Thereby the dDG$\mathcal{L}$ calculus reduces properties of hybrid games to d$\mathcal{L}$ properties of hybrid programs, which it, in turn, reduces to validity questions of formulas in first-order logic over the reals like the d$\mathcal{L}$ proof calculus does [Pla08].

The first sequent calculus was developed in 1935 by Gerhard Gentzen and is a calculus for first-order logic [Gen35]. The central idea of a sequent calculus is to split the existing formulas into atomic ones and separate those atomic formulas which are *true* from those which are *false*. The calculus constructs a direct proof for the validity of a formula.

A sequent is an implication between the conjunction of the formulas on the left side and the disjunction of the formulas from the right side.

Definition 45 (Sequent)**.** $\Gamma \vdash \Delta$ *is a* sequent *with sets of formulas* Γ *and* Δ*.*

It is satisfiable *if for some state* ν *it holds* $\nu \not\models \varphi$ *for some formula* $\varphi \in \Gamma$ *or* $\nu \models \psi$ *for some formula* $\psi \in \Delta$*.*

The sequent is valid*, if for all states* ν *it holds* $\nu \not\models \varphi$ *for some* $\varphi \in \Gamma$ *or* $\nu \models \psi$ *for some formula* $\psi \in \Delta$*.*

Proof rules are applied from the desired conclusion (goal below bar) to the resulting premises (above bar) that need to be proved instead.

Definition 46 (Rules [Pla10b])**.** Calculus rules *are defined from the rule schemata presented in Figure 4.6, Figure 4.7, and Figure 4.8 using the following definitions:*

1. *If*

$$\frac{\phi_1 \vdash \psi_1 \quad \dots \quad \phi_n \vdash \psi_n}{\phi_0 \vdash \psi_0}$$

is an instance of a rule schema in Figure 4.6, Figure 4.7, or Figure 4.8, then

$$\frac{\Gamma, \langle\!\langle \mathcal{J} \rangle\!\rangle \phi_1 \vdash \langle\!\langle \mathcal{J} \rangle\!\rangle \psi_1, \Delta \quad \dots \quad \Gamma, \langle\!\langle \mathcal{J} \rangle\!\rangle \phi_n \vdash \langle\!\langle \mathcal{J} \rangle\!\rangle \psi_n, \Delta}{\Gamma, \langle\!\langle \mathcal{J} \rangle\!\rangle \phi_0 \vdash \langle\!\langle \mathcal{J} \rangle\!\rangle \psi_0, \Delta}$$

can be applied as a proof rule, where Γ, Δ *are arbitrary (possible empty) finite sets of context formulas and* $\mathcal{J}$ *is a (possibly empty) discrete assignment in either a box or a diamond modality. We write* $\langle\!\langle \mathcal{J} \rangle\!\rangle$ *to denote either* $[\mathcal{J}]$ *or* $\langle \mathcal{J} \rangle$.

2. *Symmetric schemata*

$$\frac{\phi}{\psi}$$

can be applied on either side of the sequent as

$$\frac{\Gamma, \langle\!\langle \mathcal{J} \rangle\!\rangle \phi \vdash \Delta}{\Gamma, \langle\!\langle \mathcal{J} \rangle\!\rangle \psi \vdash \Delta} \quad \textit{or as} \quad \frac{\Gamma \vdash \langle\!\langle \mathcal{J} \rangle\!\rangle \phi, \Delta}{\Gamma \vdash \langle\!\langle \mathcal{J} \rangle\!\rangle \psi, \Delta}$$

Again they do not alter the context. Additionally, we use the abbreviation $\langle\!\langle \alpha \rangle\!\rangle$ *if the rule is independent of the player controlling the action* α.

3. *The existential quantifier elimination rule applies to all goals containing variable X at once: If* $\phi_1 \vdash \psi_1, \ldots, \phi_n \vdash \psi_n$ *is the list of all open goals (i.e., goals that have not been proved yet) of the proof that contain the free variable X, then the following instance can be applied as a proof rule:*

$$\frac{\vdash QE(\exists X \bigwedge_i (\phi_i \vdash \psi_i))}{\phi_1 \vdash \psi_1 \quad \ldots \quad \phi_n \vdash \psi_n}$$

As this definition of proof rules that we inherited from the work of Platzer on d$\mathcal{L}$ [Pla10b] contains rules that produce multiple conclusions we also repeat the definition of provability from [Pla10b].

Definition 47 (Provability [Pla10b])**.** *A* derivation *is a finite acyclic graph labeled with sequents such that, for every node, the (set of) labels of its children must be the (set of) premises of an instance of one of the calculus rules and the (set of) labels of the parents of these children must be the (set of) conclusions of that rule instance. A formula* ψ *is* provable *from a set* Φ *of formulas iff there is a finite subset* $\Phi_0 \subseteq \Phi$ *for which the sequent* $\Phi_0 \vdash \psi$ *is derivable, i.e., there is a derivation with a single root (i.e., node without parents) labeled* $\Phi_0 \vdash \psi$.

(P1) $\dfrac{\vdash A}{\neg A \vdash}$ (P6) $\dfrac{A, B \vdash}{A \wedge B \vdash}$ (P11) $\dfrac{*}{A \vdash A}$

(P2) $\dfrac{A \vdash}{\vdash \neg A}$ (P7) $\dfrac{\vdash A \quad \vdash B}{\vdash A \wedge B}$ (P12) $\dfrac{A \vdash \quad \vdash A}{\vdash}$

(P3) $\dfrac{A \vdash B}{\vdash A \to B}$ (P8) $\dfrac{\vdash A \quad B \vdash}{A \to B \vdash}$

(P4) $\dfrac{A \vdash \quad B \vdash}{A \vee B \vdash}$ (P9) $\dfrac{A, B \vdash \quad \vdash A, B}{A \leftrightarrow B \vdash}$

(P5) $\dfrac{\vdash A, B}{\vdash A \vee B}$ (P10) $\dfrac{A \vdash B \quad B \vdash A}{\vdash A \leftrightarrow B}$

- A and B are schemata of arbitrary dDGℒ formulas

Figure 4.6: Propositional rules

Propositional Rules. The rules for handling the propositional part of the formulas are illustrated in figure 4.6. In sequent calculus, the idea is to sort the formulas into a proof tree in a specific way.

The branches of the proof tree are connected by conjunctions. Thus, the rule P7 splits the proof along conjunctions on the right side of the sequent. This follows from the fact that the expression $\vdash A \wedge B$ refers to the fact that we have to show that A and B hold in the current context. Similarly, the rule P4 splits the proof along disjunctions in the antecedent of the sequent as $A \vee B \vdash$ refers to the fact that we have to show that our conclusion follows from A and from B independently. On the same line is the rule P8 as an implication again basically is the same as an disjunction $A \to B \equiv \neg A \vee B$.

The equivalence is handled by the rules P9 and P10. Both rules split the proof into two branches. For the case, that the equivalence occurs on the left side, it is necessary to show that the proof can be closed if the equivalence is valid, i.e., either both sides are true or both sides are false. If

the equivalence would be false the sequent would trivially be valid. For the other case, we have to show that the formulas connected by the equivalence are complementary. This causes a branch as well, as again there are two cases to consider.

Having understood the case where the proof is split into multiple branches, we now turn our focus to rules that work on a single branch.

The rules P1 and P2 eliminate negations. A negation in a sequent can be eliminated by moving the formula from one side to the other. To show that a sequent is satisfied, we have to show that there is a formula on the left side that is false or a formula on the right side that is true. If e.g. $\neg A$ occurs on the right side of the sequent, it is obvious that we can add A on the left side of the sequent, because if we can show that A is false, we have also shown that $\neg A$ is true.

The rules P6, P5 and P3 eliminate conjunctions, disjunctions and implication in the cases where no branch is necessary. A sequent is by definition an implication between the conjunction of the formulas on the left side and the disjunction of the formulas from the right side. As formulas on the left side are connected implicitly by a conjunction, we can remove conjunctions here. The same holds for the implicit disjunction and disjunctions on the right side. In total the sequent represents an implication. So if an implication occurs on the right side, we can move its premise to the left side and just keep its conclusion on the right side of the sequent.

For closure of the proof the rule P11 is used. It can be applied if there is the same formula occurring on both sides of the sequent. A rule for closure is a rule without premises, thus it leads to a leaf of the proof graph.

Additionally, the calculus features a cut rule P12. This rule allows us to make a case distinction based on a formula A. That is the proof splits into two branches. On the one branch we show that A holds in the current context. On the other branch we show that our conclusion holds under the assumption that A is satisfied.

First-order and modality rules. The calculus performs a symbolic execution. In order to keep track of the symbolic state of the program we use a simultaneous assignment (also called discrete jump set) $\mathcal{J}$ of the form $\langle\!\langle x_1 := \theta_1, \ldots, x_n := \theta_n \rangle\!\rangle$. During our symbolic execution, whenever we execute a discrete assignment, we merge it with this jump set. Finally, once all modalities have been dealt with, we use rule D8 to get rid of this assignment by substitution. That way, we have to define substitutions only

on first-order formulas (see Definition 7 on page 24). See [Pla08, Pla10b] for more details on this matter. We refer to the variables that might by altered during playing a game G as bound variables. We denote by $\forall^G \phi$ the *universal closure* of the formula ϕ w.r.t. the variables bound in G.

Figures 4.6–4.8 show a proof calculus for dDG$\mathcal{L}$. Together with rules for dealing with propositional logic shown in Figure 4.6, the calculus rules D1-D20 form the original proof calculus for d$\mathcal{L}$ [Pla08, Pla10b]. The rules D1-D12 are equivalences for transforming and decomposing hybrid programs. Rules D1 and D2 are used to handle tests within programs. In the diamond case, we have to check that the test condition is satisfied. Otherwise, no execution of the program is possible. Hence the rule results in a conjunction of the test condition and the post condition of the modality. For the box case the situation is different. Here, the box talks about all execution of the program. Thus, if the test condition is not satisfied we do not have to show anything. Therefore, the formula is transformed into an implication where we have to show that whenever the test condition is satisfied then the post condition of the modality also holds. Rule D3 deals with sequential compositions. Here, we can just split the modality along the sequential operator as the modalities talk about reachability in the same way as this operator does. For choices we have to distinguish between the different modalities again. For the diamond case rule D4 is used. It expresses that there is a some execution of the program $\alpha \cup \beta$ that satisfies ϕ if, and only if, there is an execution of α or an execution of β that does so. For the box case, we have to ensure that both such cases lead into states that satisfy ϕ. This is expressed in rule D5. Nondeterministic assignments correspond to existential quantification in the diamond case (rule D10) and to universal quantification in the box case (rule D9). For handling continuous evolutions, we add rules that are based on the solution of the differential equations. That is, a formula ϕ is satisfied after some execution of a continuous evolution if there is some evolution time t such that for all intermediate states the evolution domain constraint is satisfied and at time t the formula ϕ holds where we replace all occurrences of the variables that are evolving by their solutions at time t. This is expressed by rule D11. For the box the only difference is that this property has to hold for all possible evolution times t instead of just some t (see rule D12).

For handling loops within programs we add two different rule types. On the one hand, the rules D6 and D7 allow for unwinding of loops. On the other hand, the rules D13 (resp. D14) allow reasoning about loops using induction (resp. proving convergence).

$$\text{(D1)}\ \frac{\phi \wedge \psi}{\langle ?\phi \rangle\, \psi}\quad \text{(D2)}\ \frac{\phi \to \psi}{[?\phi]\, \psi}\quad \text{(D3)}\ \frac{\langle\!\langle \alpha \rangle\!\rangle \langle\!\langle \beta \rangle\!\rangle\, \phi}{\langle\!\langle \alpha;\beta \rangle\!\rangle\, \phi}\quad \text{(D4)}\ \frac{\langle \alpha \rangle\, \phi \vee \langle \beta \rangle\, \phi}{\langle \alpha \cup \beta \rangle\, \phi}$$

$$\text{(D5)}\ \frac{[\alpha]\, \phi \wedge [\beta]\, \phi}{[\alpha \cup \beta]\, \phi}\quad \text{(D6)}\ \frac{\phi \vee \langle \alpha;\alpha^* \rangle\, \phi}{\langle \alpha^* \rangle\, \phi}\quad \text{(D7)}\ \frac{\phi \wedge [\alpha;\alpha^*]\, \phi}{[\alpha^*]\, \phi}$$

$$\text{(D8)}\ \frac{\phi\{x_1 \mapsto \theta_1, \ldots, x_n \mapsto \theta_n\}}{\langle\!\langle x_1 := \theta_1, \ldots, x_n := \theta_n \rangle\!\rangle\, \phi}$$

$$\text{(D9)}\ \frac{\forall t [x := t] \phi}{[x := *]\, \phi}\quad \text{(D10)}\ \frac{\exists t [x := t] \phi}{\langle x := * \rangle\, \phi}\quad \text{(D11)}\ \frac{\exists\, t \geq 0\, \forall\, 0 \leq \tilde{t} \leq t \left\langle x := y_v(\tilde{t}) \right\rangle \chi \to \langle x := y_v(t) \rangle\, \phi}{\langle \dot{x} = \theta \& \chi \rangle\, \phi}$$

$$\text{(D12)}\ \frac{\forall\, t \geq 0\, \forall\, 0 \leq \tilde{t} \leq t \left[x := y_v(\tilde{t})\right] \chi \to [x := y_v(t)]\, \phi}{[\dot{x} = \theta \& \chi]\, \phi}$$

$$\text{(D13)}\ \frac{\vdash \forall^{[\alpha]}(\phi \to [\alpha]\, \phi)}{\phi \vdash [\alpha^*]\, \phi}\quad \text{(D14)}\ \frac{\vdash \forall^{\langle \alpha \rangle} \forall n > 0\, (\varphi(n) \to \langle \alpha \rangle\, \varphi(n-1))}{\exists n \varphi(n) \vdash \langle \alpha^* \rangle\, \exists n\, (n \leq 0 \wedge \varphi(n))}$$

$$\text{(D15)}\ \frac{\vdash \phi(s(X_1, \ldots, X_n))}{\vdash \forall x\, \phi(x)}\quad \text{(D16)}\ \frac{\phi(s(X_1, \ldots, X_n)) \vdash}{\exists x\, \phi(x) \vdash}\quad \text{(D17)}\ \frac{\vdash \phi(X)}{\vdash \exists x\, \phi(x)}\quad \text{(D18)}\ \frac{\phi(X) \vdash}{\forall x\, \phi(x) \vdash}$$

$$\text{(D19)}\ \frac{\vdash \mathrm{QE}(\exists X \bigwedge_i (\phi_i \vdash \psi_i))}{\phi_1 \vdash \psi_1 \quad \ldots \quad \phi_n \vdash \psi_n}$$

$$\text{(D20)}\ \frac{\vdash \mathrm{QE}(\forall X \phi(X) \vdash \psi(X))}{\phi(s(X_1, \ldots, X_n)) \vdash \psi(s(X_1, \ldots, X_n))}$$

- t and $\tilde{t}$ are fresh logical variables, y_v is the solution of the symbolic initial value problem $(\dot{x} = \theta, x(0) = v)$.
- Logical variable n does not occur in α.
- $\phi\{x_1 \mapsto \theta_1, \ldots, x_n \mapsto \theta_n\}$ denotes the formula where each x_i is substituted by θ_i simultaneously. The assignment in rule D8 must be admissible [Pla10b], otherwise it is added to the jump context $\langle\!\langle \mathcal{J} \rangle\!\rangle$.
- X is a new logical variable.
- QE: quantifier elimination procedure (can only be applied to first-order formulas).
- For D19 $\phi_i \vdash \psi_i$ are the only branches where X occurs as a free logical variable.

Figure 4.7: Rules for first-order and dℒ-operators

$$\text{(G1)}\ \frac{G_1\phi \vee G_2\phi}{(G_1 \cup G_2)\phi} \quad \text{(G3)}\ \frac{G_1(G_2\phi)}{(G_1G_2)\phi} \quad \text{(G5)}\ \frac{\phi \vee G\ (G)^{\langle * \rangle}\phi}{(G)^{\langle * \rangle}\phi}$$

$$\text{(G2)}\ \frac{G_1\phi \wedge G_2\phi}{(G_1 \cap G_2)\phi} \quad \text{(G4)}\ \frac{\phi \wedge G\ (G)^{[*]}\phi}{(G)^{[*]}\phi} \quad \text{(G6)}\ \frac{\vdash \forall^G(\phi \to \psi)}{G\phi \vdash G\psi}$$

$$\text{(G7)}\ \frac{\vdash \forall^G(\phi \to G\phi)}{\phi \vdash (G)^{[*]}\phi} \quad \text{(G8)}\ \frac{\vdash \forall^G \forall n > 0\,(\varphi(n) \to G\,(\varphi(n-1)))}{\exists n\,\varphi(n) \vdash (G)^{\langle * \rangle}\exists n\,(n \leq 0 \wedge \varphi(n))}$$

Logical variable n does not occur G.

Figure 4.8: Proof Rules for dDGℒ-operators

For handling first-order quantifiers, we use rules D15-D20 from dℒ [Pla08]. They perform Skolemization [Pla10b] to allow for removing the modalities within the formulas using other rules. After the modalities are dealt with, the quantifiers are reintroduced and quantifier elimination (QE) is performed. Let us focus on quantifiers on the right side of the sequent. In that case, for existentially quantified variables we use free variables instead. This step itself would be sound as the resulting formula is actually strong than the original one. However, we store the information that the variable was existentially quantified in order to later en reintroduce the existential quantifier with rule D19 once all modalities below the quantifier are dealt with. We replace universally quantified variables by fresh free function symbols. As parameters to this function symbol, we add all variables that where existentially quantified before. This is sound, as the formula is only valid, if it is satisfied for all interpretation of the free function symbols. Later in the proof, we use the arguments of the Skolem symbols (i.e., those variables that are arguments of free functions) to reconstruct the quantifier order before performing quantifier elimination. The cases for quantifiers on the left side of the sequent are dual. Observe that, the leafs of our proof graph, are connected by conjunctions. Hence for the reintroduction of the existential quantifier, we have to collect all branches on which the free variable occurs because we need a common valuation for this variable. This is crucial as introducing an existential quantifier on each branch locally would yield unsound results. This can be easily seen if we recall that $(\exists x\ \phi(x) \wedge \exists x\ \psi(x))$ does not imply that there is some common x satisfying the formulas ϕ and ψ, i.e., $\exists x\ (\phi(x) \wedge \psi(x))$. This becomes obvious when considering two contradictory formulas, for instance, $\phi \equiv x = 0$ and $\psi \equiv x = 1$. The rules for the reintroduction of the universal quantifier,

however, work locally on a single branch.

Game rules. We have extended this proof calculus with additional rules for handling the game specific constructs appearing in dDG$\mathcal{L}$ formulas (rules G1-G8 in Figure 4.8).

The rules G1 and G2 are equivalences to reason about the choice operations on games. They are the game-equivalents of D5 and D4. Rule G3 transforms sequential compositions such that they can be handled the other rules. Observe that $G_1(G_2\phi)$ is structurally simples than $(G_1\,G_2)\phi$ as it does contain one operator less (the sequential composition of games). The rules G4-G5 allow for unwinding of the game loops. Rule G7 follows the pattern of D13, but allows induction over game loops that are under Falsifier's control. If Verifier can establish that a formula ϕ holds after any run of game G that started in an arbitrary state satisfying ϕ, then, by induction, ϕ holds for an arbitrary number of plays. Rule G8 follows the pattern of D14 and can be used to show properties of game loops that are under Verifier's control. We can be sure that there is a number of iterations after which the postcondition $\varphi(n)$ holds for some $n \leq 0$ if G can be controlled by Verifier such that the state converges w.r.t. $\varphi(n)$. Here, the existence of some n such that $\varphi(n)$ holds serves as an induction anchor. As for each play started in an arbitrary state where $n > 0$ and $\varphi(n)$ holds, Verifier can assure that after playing the game G the formula $\varphi(n-1)$ holds, thus the game can be forced to eventually reach a state where $n \leq 0$ and $\varphi(n)$ holds. Note that n must not occur in the game G as otherwise it would be bound by the game instead of the quantifier prefix in the postcondition and thus falsify our induction. Additionally, the generalization rule G6 can be used to strengthen postconditions. This rule can, for example, be used to add induction anchors and use cases to the rules G7 and G8.

The purpose of the calculus is to provide a framework for deriving valid dDG$\mathcal{L}$ formulas syntactically. A calculus is sound iff all formulas derived by applying the calculus rules are indeed valid.

Definition 48 (Soundness)**.** *A calculus rule* $\frac{\phi_1,\ldots,\phi_n}{\psi_1,\ldots,\psi_n}$ *is* sound *iff validity of the premises* $\phi_1 \wedge \cdots \wedge \phi_n$ *implies the validity of the conclusions* $\psi_1 \wedge \cdots \wedge \psi_n$*.*

Theorem 6. *The* dDG$\mathcal{L}$ *calculus rules presented in Figures 4.6–4.8 are sound.*

The soundness proofs for the rules P1-P12 and D1-D20 in [Pla10b] are valid for dDG$\mathcal{L}$ as well, because dDG$\mathcal{L}$ is a conservative extension of d$\mathcal{L}$ (see Theorem 5).

The soundness of the rules G1 and G2 is obvious from the semantics of the operators $\cup$ and $\cap$ on games. For the rule G3 the soundness follows directly from the definition of the sequential composition. The soundness of the unwinding rules G4 (resp. G5) is a direct consequence of the semantics of $(G)^{[*]}$ (resp. $(G)^{\langle * \rangle}$). For proving soundness of rule G6 a similar pattern to that in [Pla10b] can be applied. The game G can only change the variables that occur in G. Therefore, if $\phi \to \psi$ and $G\phi$ holds independent of how the variables occurring in G are evaluated, ψ also holds after playing G.

Soundness of the induction rule G7 and the convergence rule G8 can be shown by induction over the number of executions of the loop, in analogy to the soundness proofs for D13 and D14 [Pla10b].

Proof of Theorem 6. We prove the soundness of each rule separately.

G1 Let ν be a state such that $\nu \models G_1\phi \vee G_2\phi$. This means $\nu \models G_1\phi$ or $\nu \models G_2\phi$. This is, by Definition 34, equivalent to $\nu \models (G_1 \cup G_2)\phi$.

G2 Let ν be a state such that $\nu \models G_1\phi \wedge G_2\phi$. This means $\nu \models G_1\phi$ and $\nu \models G_2\phi$. This is, by Definition 34, equivalent to $\nu \models (G_1 \cap G_2)\phi$.

G3 Let ν be a state, $\nu \models (G_1\ G_2)\phi$ iff $\nu \models G_1(G_2\phi)$, by Definition 34.

G4 Let ν be a state such that $\nu \models \phi \wedge G((G)^{[*]}\phi)$. This means, $\nu \models \phi$ and $\nu \models G((G)^{[*]}\phi)$. The latter means that for all $m \in \mathbb{N}$ we have $\nu \models G(G^m\phi)$. Therefore, $\nu \models G^n\phi$ for all $n \in \mathbb{N}$: It holds for $n = 0$ as $\nu \models \phi$, and for $n > 0$ as $\nu \models G((G)^{[*]}\phi)$. This concludes the proof as this is equivalent to $\nu \models (G)^{[*]}\phi$ by Definition 34.

G5 Let ν be a state such that $\nu \models \phi \vee G((G)^{\langle * \rangle}\phi)$. This means, $\nu \models \phi$ or $\nu \models G((G)^{\langle * \rangle}\phi)$. Now we make a case distinction. If $\nu \models \phi$ then also, by definition of G^0, $\nu \models G^0\phi$. Therefore, there is an $n \in \mathbb{N}$ such that $\nu \models G^n\phi$ and, thus, $\nu \models (G)^{\langle * \rangle}\phi$, by Definition 34. If $\nu \models G((G)^{\langle * \rangle}\phi)$, then there is $m \in \mathbb{N}$ such that $\nu \models G(G^m\phi)$. Therefore, $\nu \models G^{m+1}\phi$ for some $m \in \mathbb{N}$. This also means that $\nu \models G^n\phi$ for some $n \in \mathbb{N}$. This concludes the proof as this is equivalent to $\nu \models (G)^{\langle * \rangle}\phi$ by Definition 34.

G6 Let ν be a state such that $\nu \models \forall^G(\phi \to \psi)$ and $\nu \models G\phi$. The only variables that can change during game G are those that are bound variables in G. Since the universal closure $\forall^G$ quantifies all those variables universally, we know that, no matter what values they take, the implication $\phi \to \psi$ holds, so we can conclude that $\nu \models G\psi$.

G7 Let ν be a state such that $\nu \models \forall^G(\phi \to G\phi)$ and $\nu \models \phi$. Consider any state ω that we reach by playing G. Then ω and ν only differ with respect to the bound variables in G. No matter what value these bound variables have (universally quantified by $\forall^G$), playing G preserves ϕ. Thus, we can conclude that $\omega \models \phi$. By induction, this gives that $\nu \models G^n\phi$ for all $n \in \mathbb{N}$ which implies $\nu \models (G)^{[*]}\phi$ by Definition 34.

G8 Let ν be a state. Assume $\nu \models \forall^G \forall n > 0\,(\varphi(n) \to G\,(\varphi(n-1)))$ and that $\nu \models \exists n\,\varphi(n)$. The latter gives us that there is a number r such that $\omega \models \varphi$ holds where ω coincides with ν upto the valuation of n which is r. By induction on r, we show that $\nu \models (G)^{\langle * \rangle}\exists n\,(n \leq 0 \wedge \varphi(n))$. If $r \leq 0$ that post condition already holds and iterating G zero times is sufficient. Otherwise, if $r > 0$, we know from $\nu \models \forall^G \forall n > 0\,(\varphi(n) \to G\,(\varphi(n-1)))$ that independent of the initial valuations of the variables altered while playing G the formula φ holds after one iteration with the altered valuation $n \mapsto r-1$. As this yields a strictly monotonically decreasing function with progress ≥ 1, the induction is well founded and we can conclude that there is a number of iteration of G after which we reach a state such that $\exists n\,(n \leq 0 \wedge \varphi(n))$ holds. Therefore, $\nu \models (G)^{\langle * \rangle}\exists n\,(n \leq 0 \wedge \varphi(n))$. □

Derived rules. In practice we consider it is useful to have some derived rules for handling loops [Pla10b]. For hybrid games two such rules can be obtained by obtained by combining rules G6 with G7 as well as rules G6 with G8. We depicted the resulting rules in Figure 4.9. Rule G9 allows to prove that in the current context after arbitrary iterations of the loop the formula ψ holds. For this, we need to find an inductive invariant ϕ that holds in the current state, is preserved by the loop and implies ψ in every state that might be reachable by executing the loop. For this, we use universal quantification $\forall^G$ that binds all variables that might be changed in the game G and, thus, only keeps information about variables that are constant w.r.t. to G from the current context. Thus, whatever state is reached by playing G the implication $\phi \to \psi$ holds. In order to prove that a formula ψ holds after some number of iterations of a game G we can use

rule G10. Here, we have to find a variant $\varphi(n)$ that serves as a termination function. We first have to prove that it holds for some n in the current state. Subsequently, we have to show that by playing G we make progress that can be measured by φ. That is started in some state with an arbitrary valuation of the variables that are bound by G and some $n > 0$, if $\varphi(n)$ holds, then Verifier has a strategy such that by playing G once we reach a state where $\varphi(n-1)$ holds. This gives a descending chain and the only thing left to show is that whenever there is some $n \leq 0$ such that $\varphi(n)$ holds then ψ holds as well. The same pattern can be used to get useful

$$\text{(G9)}\ \frac{\vdash \phi \quad \vdash \forall^G(\phi \to G\phi) \quad \vdash \forall^G(\phi \to \psi)}{\vdash (G)^{[*]}\psi}$$

$$\text{(G10)}\ \frac{\begin{array}{l}\vdash \exists n\, \varphi(n)\\ \vdash \forall^G \forall n > 0\, (\varphi(n) \to G\, (\varphi(n-1)))\\ \vdash \forall^G(\exists n(n \leq 0 \wedge \varphi(n)) \to \psi)\end{array}}{\vdash (G)^{\langle * \rangle}\psi}$$

Logical variable n does not occur G.

Figure 4.9: Derived proof rules for dDGℒ-operators

derived rules for hybrid programs (see Figure 4.10). The proofs that these

$$\text{(D11)}\ \frac{\vdash \phi \quad \vdash \forall^{[\alpha]}(\phi \to [\alpha]\phi) \quad \vdash \forall^{[\alpha]}(\phi \to \psi)}{\vdash [\alpha^*]\psi}$$

$$\text{(D12)}\ \frac{\begin{array}{l}\vdash \exists n \varphi(n)\\ \vdash \forall^{\langle\alpha\rangle} \forall n > 0 (\varphi(n) \to \langle\alpha\rangle(\varphi(n-1)))\\ \vdash \forall^{\langle\alpha\rangle}(\exists n(n \leq 0 \wedge \varphi(n)) \to \psi)\end{array}}{\vdash \langle\alpha^*\rangle\psi}$$

Logical variable n does not occur α.

Figure 4.10: Derived proof rules for dℒ-operators

rules are derived are all similar and we thus decided to just show the proof for G9 as an example in Figure 4.11.

$$\dfrac{\dfrac{\dfrac{\vdash \forall^G(\phi \to (G)^{[*]}\phi)}{\phi \vdash (G)^{[*]}\phi}\,\text{G7}}{\vdash \phi \to (G)^{[*]}\phi}\,\text{P3} \qquad \dfrac{\vdash \phi \qquad \dfrac{\vdash \forall^G(\phi \to \psi)}{(G)^{[*]}\phi \vdash (G)^{[*]}\psi}\,\text{G6}}{\phi \to (G)^{[*]}\phi \vdash (G)^{[*]}\psi}\,\text{P8}}{\vdash (G)^{[*]}\psi}\,\text{P12}$$

Figure 4.11: Proof that Rule G9 is a derived rule

4.4 Case Study: Robotic Factory Automation

To demonstrate the applicability of our approach we model a factory automation scenario in which an autonomous robot moves in an automatic factory. For scalability reasons, central coordination and planning become infeasible, so the factory is set up as a collection of autonomous agents pursuing goals that may not be known globally. The robot has a secondary objective of reaching certain target positions, but its primary objective is to stay safe, i.e., neither leave the factory site nor bump into its surrounding wall, which could damage the robot.

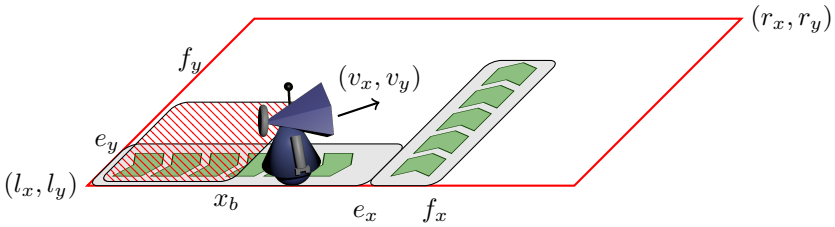


Figure 4.12: Sketch of the robotic factory automation site

Model. We model a robot with position (x, y), velocity $\vec{v} = (v_x, v_y)$, and acceleration $\vec{a} = (a_x, a_y)$ on a 2 dimensional rectangular factory ground (Figure 4.12). There are two conveyor belts. One pointing in x-direction and one pointing in y-direction. The factory may independently decide to activate the conveyor belts, in which case they increase the velocity of the robot. The robot may decide to move in any direction. Therefore, it can

decelerate to try to compensate for this increased speed. The goal of the robot is to avoid crashing into any wall and avoid other machines using the belt.

A sketch of the factory site is provided in Figure 4.12. One conveyor belt is of y-width e_y between positions l_x and e_x and moves in x-direction if activated. Between e_x and f_x there is a belt of y-width f_y moving in y-direction. The shaded region in Figure 4.12 indicates a region that has to be cleared within ε time units after the system was started, because other robotic elements of the factory may occupy this space then and not watch out for our robot. For simplicity, the robot is initially located at the lower left end (l_x, l_y) of the factory site. The conveyor belt in x-direction has a maximal velocity of c_x and that in y-direction of c_y. The conveyor belts accelerate very quickly, so we simply consider them to accelerate instantaneously. Thus, upon activation, their effect is to increase the velocity of the robot by a discrete assignment instantaneously if the robot is currently located on the conveyor belt that got activated. The robot itself can accelerate with any acceleration of absolute value at most $A = 2$ and that acceleration can be applied in x-direction (acceleration a_x) or y-direction (acceleration a_y) or both. The robot can activate a brake that will slow it down. The difference between braking and just accelerating in the opposite direction is that braking does not allow changes of the sign of the velocity but instead stops at velocity 0.

Specification. As the robot is a moving object and cannot come to a standstill instantaneously, certain conditions have to be satisfied to allow safe operation. Therefore, we assume the following conditions on the scenario. We require that the point x_b can be reached by accelerating for at most time ε, the x-belt moves to the right (if activated), and after passing the belts there is enough space to brake from the velocity we reach by accelerating for four cycles (each of duration $\leq \varepsilon$) and possible extra velocity gained when a conveyor belt activates:

$$x_b < \frac{1}{2}A\varepsilon^2 \wedge c_x > 0 \wedge (c_x + 4A\varepsilon)^2 \leq 2A(r_x - f_x) \tag{4.1}$$

For the y-direction we assume

$$c_y > 0 \wedge c_y^2 \leq 2A(r_y - l_y) \ , \tag{4.2}$$

i.e., the y-belt moves upwards (if activated) and there is enough space for the robot to compensate for the effects of the conveyor belt by braking long enough without having to leave the factory ground.

Even though these constraints limit the possible scenarios, we haven proven that a strategy for the robot exists such that it meets its objectives. Figure 4.13 shows the hybrid game describing the robotic factory scenario. The game is structured as follows. First the environment, i.e., Falsifier, chooses a number of iterations for the loop shown in lines 1-10. In each iteration, the environment may choose to activate one of the conveyor belts if the robot is on it. This is modeled with the modality in lines 1-2. Here, the variable eff_1 (resp. eff_2) is used in order to ensure that the conveyor belts are activated at most once. These variables is initialized with the value 1 and set to 0 once the action is executed. As the branch first tests whether the robot is on the conveyor belt and the variable value is 1, the action can only be execute once. Afterwards, the robot (i.e., Verifier) chooses his accelerations in x and y-direction (line 3). The clock t_s is reset (also line 3) to measure the cycle time (i.e., $t_s \leq \varepsilon$), then the robot chooses (line 6) if it wants to brake (line 7-8) or possibly to drive backwards w.r.t. to its current direction (line 5). The time for the continuous evolutions in lines 5 and 8 is then chosen by the environment within the cycle time constraint (for both line 5 and 8) and the zero crossing of one of the velocities (only line 8). Thus, accelerating for ε time units can take many iterations of the loop as ε only provides an upper bound on the cycle time. Further, note that for the braking case if the velocity in a direction is 0 then that acceleration is set to 0 as well to avoid time deadlocks. This is checked in line 7. Also the robot has to ensure that its choices for the acceleration are compatible with the current velocities: for a velocity v and an acceleration a, if the robot wants to brake, i.e., reduce the velocity to 0, then the product va has to be non-positive.

The winning condition for the robot is to stay safe, i.e.,

$$l_x \leq x \leq r_x \wedge l_y \leq y \leq r_y \wedge (t \geq \varepsilon \rightarrow (x \geq x_b \vee y \geq e_y)) \ .$$

The robot must stay within the rectangle of the points (l_x, l_y) and (r_x, r_y) but has to leave the rectangle (l_x, l_y) and (x_b, e_y) after ε time units. The latter requirement models that uncooperative robotic elements might enter that region. Note that the number of iterations is chosen when the game starts not when specifying the system. Sensor and communication delays are not modeled explicitly here. Since they are beyond control for the

robot, the number of iterations and the evolution durations are chosen by the factory environment. How long the robot needs to work in the factory is also decided by the factory, so the robot needs to guarantee safety for all times. However, whether the robot actually has a strategy is quite subtle. Simple strategies like always accelerating, or always braking are bound to fail and accelerating for exactly ε time units is not possible as the environment determines the actual cycle time and might not allow changing the acceleration at that exact point in time. The robot has to navigate the factory very carefully, react to changes in the conveyor belt activation as needed, and robustly adapt to the number of control loop repetitions and (possibly erratic) cycle durations chosen by the factory environment.

$$
\begin{array}{l}
1\quad \Big(\big([?true] \cap [(?(x < e_x \wedge y < e_y \wedge \mathrm{eff}_1 = 1);\ v_x := v_x + c_x;\ \mathrm{eff}_1 := 0)] \\
2\qquad\quad \cap [(?(e_x \leq x \wedge y \leq f_y \wedge \mathrm{eff}_2 = 1);\ v_y := v_y + c_y;\ \mathrm{eff}_2 := 0)]\big) \\
3\quad \langle\ a_x := *;\ ?(-A \leq a_x \leq A);\ a_y := *;\ ?(-A \leq a_y \leq A);\ t_s := 0\ \rangle \\
4\quad (\\
5\qquad \big[\dot{x} = v_x, \dot{y} = v_y, \dot{v_x} = a_x, \dot{v_y} = a_y, \dot{t} = 1, \dot{t_s} = 1 \& t_s \leq \varepsilon\big] \\
6\quad \cup \\
7\qquad (\langle ?a_x v_x \leq 0 \wedge a_y v_y \leq 0; \\
\qquad\quad \text{if } v_x = 0 \text{ then } a_x := 0 \text{ fi; if } v_y = 0 \text{ then } a_y := 0 \text{ fi}\rangle \\
8\qquad \big[\dot{x} = v_x, \dot{y} = v_y, \dot{v_x} = a_x, \dot{v_y} = a_y, \dot{t} = 1, \dot{t_s} = 1 \\
\qquad\quad \& t_s \leq \varepsilon \wedge a_x v_x \leq 0 \wedge a_y v_y \leq 0]) \\
9\quad) \\
10\quad \Big)^{[*]}
\end{array}
$$

Figure 4.13: Description of game for robotic factory automation scenario (RF)

Verification. We consider an instance of the case study that is parametric w.r.t. ε, c_x, c_y, and x_b, but we fix $l_x = l_y = 0$, $r_x = r_y = 10$, $e_x = 2$, $e_y = 1$, $f_x = 3$, $f_y = 10$. We have verified the following propositions using KeYmaera [PQ08a], to which we added dDGℒ proof rules. To establish the desired property, we first show that the robot can stay within the factory site whatever the factory does.

Proposition 6. *The following* dDGℒ *formula is valid, i.e., there is a strategy for Verifier in the game depicted in Figure 4.13 that achieves the postcondition:*

$$(x = y = 0 \wedge (4.1) \wedge (4.2)) \rightarrow (RF)(l_x \leq x \leq r_x \wedge l_y \leq y \leq r_y)$$

When proving this property, we focus on the case where the robot is not driving towards the lower left corner, i.e., negative accelerations for both the x- and y-component; see Figure 4.12.

Again allowing for arbitrary movement in x-direction, we analyze, for a projection to the x-axis (cf. Figure 4.14), a more complex postcondition, where the robot has to leave the shaded region but stay inside the factory site. Observe, that the model in Figure 4.14 is identical to that in Figure 4.13 except that actions that only influence the y-position or the velocity in y-direction have been removed.

$$
\begin{array}{l}
1 \quad \Big(\big([?true] \cap [(?(x < e_x \wedge y < e_y \wedge \mathrm{eff}_1 = 1);\ v_x := v_x + c_x;\ \mathrm{eff}_1 := 0)]\big) \\
2 \quad \langle\ a_x := *;\ ?(-A \leq a_x \leq A);\ t_s := 0\ \rangle \\
3 \quad (\\
4 \quad\quad \big[\dot{x} = v_x, \dot{v_x} = a_x, \dot{t} = 1, \dot{t_s} = 1 \& t_s \leq \varepsilon\big] \\
5 \quad \cup \\
6 \quad\quad (\langle ?a_x v_x \leq 0;\ \text{if } v_x = 0 \text{ then } a_x := 0 \text{ fi}\rangle \\
7 \quad\quad \big[\dot{x} = v_x, \dot{v_x} = a_x, \dot{t} = 1, \dot{t_s} = 1 \& t_s \leq \varepsilon \wedge a_x v_x \leq 0\big]) \\
8 \quad) \\
9 \quad \Big)^{[*]}
\end{array}
$$

Figure 4.14: Projection of robotic factory automation scenario ($RF|_x$)

Proposition 7. *The following* dDGℒ *formula is valid, i.e., there is a strategy for Verifier in the game in Figure 4.13 projected to the x-axis (denoted $RF|_x$) that achieves the postcondition:*

$$(x = y = 0 \wedge (4.1)) \rightarrow (RF|_x)(l_x \leq x \leq r_x \wedge (t \geq \varepsilon \rightarrow (x \geq x_b)))$$

In the proof of Proposition 7 we show that the following invariant is an inductive invariant:

$$\begin{aligned}
&\text{eff}_1 \in \{0,1\} \wedge x \geq l_x \wedge v_x \geq 0 \wedge (t \geq \varepsilon \rightarrow x \geq x_b) \\
&\wedge (v_x + c_x\text{eff}_1)^2 \leq 2A(r_x - x) \\
&\wedge \Big(x < x_b \rightarrow t \leq \varepsilon \wedge \big(x_b - x \leq \frac{1}{2}A\varepsilon^2 - \frac{1}{2}At^2 \\
&\wedge (\text{eff}_1 = 1 \rightarrow v_x = At) \wedge (\text{eff}_1 = 0 \rightarrow v_x = At + c_x) \\
&\wedge r_x - x \geq \frac{(v_x + \text{eff}_1 c_x)^2}{2A} + A(2\varepsilon - t)^2 + 2(2\varepsilon - t)(v_x + \text{eff}_1 c_x)\big)\Big) \quad (4.3)
\end{aligned}$$

The invariant says that enough space remains to brake before reaching the right end of the factory ground. Additionally, if the point x_b has not yet been passed then the time is not up and the distance to the right wall is bounded by the space the robot can cover by accelerating ε time units and the distance it could already have covered within the current runtime. Further, the distance to the far right side is large enough to accelerate for another $2\varepsilon - t$ time units and brake afterwards without hitting the wall. The 2ε time units are necessary as Falsifier chooses how long to evolve and the robot may accelerate for ε time units before it can react again. Therefore, the robot may have to accelerate when clock t is almost ε and then may not react again within the next ε time units.

The KeYmaera proof for Proposition 6 has 2471 proof steps on 742 branches; 159 steps were performed interactively. The proof for Proposition 7 has 375079 proof steps on 10641 branches (1673 interactive steps). The interactive steps provide the invariant and simplify the resulting arithmetic. Note that Proposition 6 is significantly simpler than Proposition 7, because there is a simpler strategy that ensures safety (Proposition 6), whereas the dDGℒ formula in Proposition 7 is only valid when the robot follows a subtle strategy to leave the shaded region quickly enough without picking up too much speed to get itself into trouble when conveyor belts decide to activate. Specifically, Proposition 7 needs the much more complicated invariant (4.3). Also, the a priori restriction (and thus strategy choice) to the case where the robot is driving in the direction towards larger x and larger y values reduces the proof for Proposition 6 significantly.

As every strategy witnessing Proposition 7 is compatible with some strategy witnessing Proposition 6, we claim that the robot meets its requirements.

Automation. The base theory of real arithmetic is decidable and, therefore, supplying an invariant is in principle sufficient to verify the safety property. However, the computational complexity poses an issue in practice. To compensate for this effect we suggest a heuristic for instantiating the existential quantifiers resulting from choices of Verifier. The heuristic is based on finding maximal and minimal instantiations of the tests occurring in the game after the existential quantifier. Additionally, we run into issues with lots of unrelated formulas for the current branch. Due to the large number of interactions between the players there are a lot of formulas lurking around providing information about the system behavior that is not necessary on the current branch. To reduce the complexity for the decision procedure we try to identify these formulas and restrict the application of the quantifier elimination procedure to those formulas necessary to prove validity of the current goal. For this, we analyze the formulas on the right side of the sequent and remove constraints that do not involve these variables. Then we test for counter-examples. If there are counter-examples, we add the formulas again one by one until we end up with a counter-example free set of formulas or the complete set of formulas.

4.5 Related Work

Fairly restricted classes of hybrid games have been shown to be decidable (see e.g. [HHM99, BBC09, VPVD11]), but the general case is undecidable. Bouyer et al. [BBC09] showed that optimal reachability in o-minimal hybrid games is decidable. In their model, strategies have costs and the goal is to compute a cost optimal strategy. O-minimal hybrid systems and games require strong resets. That is, on every transition every variable is reset independent on the system state when the transition was taken. Therefore, at every jump the system "forgets" about the details of the evolution in the last mode. This either restricts the guards to equalities or makes it impossible to model continuous trajectories over several modes, i.e., to keep track of the position of the robot in our example. Hence they do not suit our needs.

Vladimerou et al. [VPVD11] extended o-minimal hybrid games to so called STORMED hybrid games and proved decidability of optimal reachability. These hybrid games are based on STORMED hybrid systems, which require that all system actions point towards a common direction (some vector $\phi \in M^n$). The M here is the set of the underlying o-minimal theory.

The O in STORMED denotes that guards, resets, and flows are o-minimal definable. The S denotes that guards are time separable. That is whenever any guard in the system is satisfied at a point $x \in M^n$ then no other guard is satisfied for any $y \in \{z \mid ||x - z|| < d\}$ for some d. This means that at any point in time at most one transition is enabled. The T denotes that the flows are time-independent which is no real restricting as every flow that is described by a differential equations has this property. The resets R are monotonic along some vector ϕ and so are the flows (M). The last property (ED) restricts the possible guards w.r.t. the common direction ϕ. That is, for each guard there has to be numbers $a, b \in M$ such that the guard is only satisfied within the set $\{\phi \cdot x \mid x \in [a, b]\}$. Unfortunately, neither STORMED hybrid games nor their special case of o-minimal hybrid games are expressive enough for our needs. Our factory automation scenario is not STORMED, for example. The issue is that we cannot find a vector ϕ such that the above requirements are satisfied. When a conveyor belt is activated there is a mode switch. Therefore, the velocity component has to be non-zero in the vector ϕ. As choices for the acceleration can lead to flows where the velocity is decreasing and some where the velocity is increasing, there is no vector ϕ with a non-zero value at the v-component satisfying the monotonicity condition on flows. See [VPVD11] for the details on the definition of STORMED hybrid games.

In these automata-based approaches, a precedence for player actions is often encoded into the semantics of hybrid game automata, e.g., controller actions have precedence over environment actions in [VPVD11]. In contrast to that dDG$\mathcal{L}$ offers more flexibility in modeling these syntactically for the particular needs of the application.

Tomlin et al. [TLS00] present an algorithm to compute maximal controlled invariants for hybrid games with continuous inputs. The class of games they consider is more general than ours as they allow inputs to differential equations to be controlled by both players, thereby added a differential game component. However, the general class of games they consider is so large that the algorithm presented is semi-decidable only for certain classes of systems, e.g., systems specified as timed or linear hybrid automata, or o-minimal hybrid systems. They further present numerical techniques to compute approximations of their reach set computation operators. However, these sometimes give unsound results. Additionally, it only works for differentiable value functions. Extending these ideas, Gao et al. [GLQ07] present a different technique for the same approach. The

drawback is that the players can neither force discrete transitions to happen nor influence which location is reached by a discrete transition.

Our approach to hybrid games has some resemblance to Parikh's propositional game logic (GL) [Par85] for propositional games. In GL in addition to the usual program constructs there is a specific construct that denotes the dual of a program. Thus, the "programs" actually become games. We call the players Angel and Demon. Usually all choices are made by Angel, only in the dual of a game the choice falls to Demon. Let W be a set of worlds. For a set of states $X \subseteq W$ the semantics denotes the set of prestates from which Angel has a strategy to ensure that the game will end in X. Thus, for a game α Angel has a strategy in the dual game α^d to ensure X from all states where it does not have strategy in α to ensure $W \setminus X$. Otherwise, Demon could use exactly this strategy to drive the game to a state outside of X. Furthermore, as the number of loop iterations is not determined a priori the semantics of loops corresponds to a fixed point computation. As an extension of propositional dynamic logic (PDL) [FL79, Pra76] the states are identified by the propositions that are satisfied in these states. In contrast to that in dDG$\mathcal{L}$ the states are valuations of real variables. Pauly and Parikh [PP03] showed that GL is strictly more expressive than PDL. Our logic dDG$\mathcal{L}$ is as a conservative extension at least as expressive as d$\mathcal{L}$. However, it is left for future work if they coincide w.r.t. expressiveness. We refer to [Pla12b] for an identification of the fundamental commonalities and differences of GL versus d$\mathcal{L}$. The distribution axiom (K) and Gödel's generalization rule generally stop to hold, for example. The classical modal axiom K is given by the follow formula:

$$[\alpha](\phi \to \psi) \to ([\alpha]\phi \to [\alpha]\psi) \tag{K}$$

As this axiom holds for d$\mathcal{L}$ it is also valid by Theorem 5 for dDG$\mathcal{L}$ as we have carefully designed the logic in order to preserve this axiom. Observe that, this means that ϕ and ψ can be dDG$\mathcal{L}$ formulas. Note however that the following more general axiom is not valid in dDG$\mathcal{L}$:

$$G(\phi \to \psi) \to (G\phi \to G\psi) \tag{KG}$$

This can be easily seen by the following example.

Example 16. *Let*

$$G \hat{=} \langle x := 0 \cup x := 1 \rangle \ ,$$
$$\phi \hat{=} x = 0 \ , \textit{ and}$$
$$\psi \hat{=} x = 2 \ .$$

In this example there is a play such that $x = 0 \rightarrow x = 2$ is satisfied. Such a state can be reached by choosing the alternative where x is set to 1, thus invalidating the premise of the implication. However, still, we can reach a state where $x = 0$ by choosing the first alternative in the program. None of these alternatives, however, leads to a state where $x = 2$ holds and thus the formula KG cannot be valid in dDG$\mathcal{L}$.

Gödel's generalization rule (often also referred to as necessitation rule or simply N) is the following:

$$\frac{\phi}{[\alpha]\phi} \tag{N}$$

Again it is sound for dDG$\mathcal{L}$ because it is so for d$\mathcal{L}$ and thus allows for the use of dDG$\mathcal{L}$ formulas for ϕ. However, note that this does not hold when we would use games instead of box modalities here. As obviously for the game $\langle ?false \rangle$ the formula $\langle ?false \rangle true$ is, unsurprisingly, a counter example to the assumption that the rule would be sound for arbitrary games.

An axiom related to K is the induction axiom. Valid for d$\mathcal{L}$ and thus dDG$\mathcal{L}$ is the following version

$$[\alpha^*](\phi \rightarrow [\alpha]\phi) \rightarrow (\phi \rightarrow [\alpha^*]\phi) \tag{I}$$

This axiom however does again not generalize to games as can be seen by the following example.

Example 17. *Let*

$$G \hat{=} \langle (x := 1 \cup (?a = 1; x := 0)); \ a := 0 \rangle \ ,$$
$$\phi \hat{=} x = 0 \ .$$

The induction axiom for games would be the following formula

$$(G)^{[*]}(\phi \rightarrow G\phi) \rightarrow (\phi \rightarrow (G)^{[*]}\phi) \ . \tag{IG}$$

However in our example, we can easily see that for a state where $a = 1$ *and* $x = 0$ *hold initially we can satisfy the premise* $(G)^{[*]}(x = 0 \to G(x = 0))$ *by choosing the branch that sets* $x := 1$ *if there is at least one iteration of the loop and otherwise the other branch to satisfy the conclusion of this implication. For the conclusion* $(x = 0 \to (G)^{[*]}(x = 0))$ *we know that* $x = 0$ *holds and thus* $(G)^{[*]}(x = 0)$ *would have to hold. However, for unwindings of the loop of length at least two we get that afterwards the* $x = 1$ *and thus* $x = 0$ *does not hold. Hence, the induction axiom IG does not hold for* dDGℒ*.*

Platzer [Pla12b] recently lifted Parikh's game logic (GL) [Par85] to hybrid systems creating differential game logic dGℒ. Like for GL and in contrast to dDGℒ the loop semantics in dGℒ does not follow the advance notice semantics. Platzer [Pla13] showed that his axiomatization of dGℒ is relatively complete w.r.t. the modal μ-calculus of differential equations [Pla13]. However, the logic dGℒ is not a conservative extension to dℒ and for example the classical distribution axiom K does not hold in dGℒ. This is a major drawback w.r.t. using our theorem prover KeYmaera as a lot of the existing rules would have to be changed. The logic presented here, dDGℒ, on the other hand can be directly implemented in KeYmaera by adding the rules presented in Figure 4.8. Furthermore, Platzer's game logic dGℒ does not feature a variant rule like we have established for dDGℒ with rule G8. Still, it is yet to be determined if the two logics actually differ in expressiveness. As the advance notice semantics of loops can be encoded in dGℒ we conjecture that every dDGℒ formula can be expressed in dGℒ. However, there is no obvious encoding in the other direction. Therefore, we assume that it would be necessary to consider the models, i.e., sets of states, that are distinguishable via dGℒ formulas and those that are distinguishable via dDGℒ formulas in order to settle this issue.

We like to note that for dGℒ it would be possible to retain the distribution axiom under additional side conditions like dual-freeness. Thus, it would be interesting to study whether a logic semantically equivalent to dGℒ but syntactically closer to dDGℒ can be conveniently defined. That is, a logic that syntactically distinguishes games and dual-free games, i.e., programs that features fixed point semantics for loops in games would be certainly worthwhile to look for.

4.6 Conclusion

In this section, we presented a game extension to differential dynamic logic (dℒ), called *differential dynamic game logic* (dDGℒ) in order to reason about hybrid games. dDGℒ is a conservative extension to dℒ. We proposed an extension to the sequent proof calculus for dℒ to handle the dDGℒ specifics. Subsequently, we applied dDGℒ as specification language for a robot factory case study.

The presented calculus is sound and inherently incomplete. Relative completeness w.r.t. to dℒ is an open question. GL [PP03] is strictly more expressive than propositional dynamic logic (PDL) [FL79, Pra76]. Witness to this is a formula that expresses the absence of an infinite branch. However, dDGℒ is able to talk about arbitrarily long computations but not about infinite ones. Therefore, this property is not expressible in dDGℒ and can thus not serve as a distinguishing example between dℒ and dDGℒ. Furthermore, unlike in PDL the computations are not part of the model anyway.

Observe that the extension from dℒ to dDGℒ was not specific to the choice of the underlying dynamic logic. That is we can perform the same construction to construct a game logic based on differential algebraic logic (DAL) [Pla10a], quantified differential dynamic logic (Qdℒ) [Pla10c] or even temporal differential dynamic logic (dTL) [Pla07c]. The first two extensions are straight forward but provide useful additional tools. DAL features more complex continuous evolutions whereas Qdℒ extends dℒ in order to reason about dynamically reconfigurable systems with arbitrary many agents. Further, it is possible to take any dynamic logic and "gamify" it using the approach presented in this chapter. This is an important strength of our approach as we leave the underlying modal logic intact and add the game operators, rules, and axioms on top of an existing theory. This, especially made it possible to easily extend our theorem prover KeYmaera in order to handle dDGℒ.

CHAPTER FIVE

Similarity and Games

Cat: Where are you going?
Alice: Which way should I go?
Cat: That depends on where you are going.
Alice: I don't know.
Cat: Then it doesn't matter which way you go.

— Lewis Carroll

Contents

In the previous chapters we introduced notions of robust refinement and a logic together with a proof calculus to prove properties of hybrid games. In this chapter, we link those two results by presenting a construction that reduces the question of refinement to the existence of a winning strategy in a hybrid game. We start by defining an alternative notion of hybrid games based on hybrid automata. Subsequently, we present a special product construction for hybrid automata and show that those can—under some restrictions—be used to serve as a witness for the fact that two systems are in refinement relation. Next, we do a similar construction on hybrid programs and propose an approach using dDG$\mathcal{L}$, our logic defined in the previous chapter, to show ε-δ-refinement of parametric hybrid systems.

Contributions. In this chapter we present two approaches to show robust refinement between system specifications. The first approach is based on hybrid automata product and model checking. The second approach builds upon our new logic dDG$\mathcal{L}$. This approach takes advantage of the special design of that language in order to allow for proving parametric robust refinement relations between hybrid programs.

Structure of the Chapter. In Section 5.1 we present a special product of automata in order to show that systems are in refinement relation. In Section 5.2 we present an approach using dDG$\mathcal{L}$ to prove refinement. We present a small example of similar systems in Section 5.2.4. We conclude this chapter in Section 5.4.

5.1 Hybrid Game Automata

The question we like to solve is how to prove that a system A robustly refines another system B. In [QFD11] we have shown that under certain restrictions it is possible to construct a hybrid game automaton such that the existence of a winning strategy for the player controlling B ensures that A robustly refines B. In this section we review these results.

5.1.1 Automata and Games

Based on hybrid automata [ACHH92, NOSY92] (see Chapter 2 for a formal semantics of hybrid automata) we define a notion of two-player hybrid game automata. For easier reference we assign names to the players: Demon and

Angel. The goal of Demon will be to drive the system into a distinctive "bad" state, whereas Angel has the goal to avoid this state. Observe, that we do not use Falsifier and Verifier here in order to make a clear distinction to dDG$\mathcal{L}$. In contrast to dDG$\mathcal{L}$ the goals of Demon and Angel will be stated in terms of reachability of a location not satisfiability of a formula. Also the possible actions of the players are different from dDG$\mathcal{L}$. That is, we allow for continuous inputs under control of Angel. Additionally, choices of Demon always have precedence over those of Angel, whereas in dDG$\mathcal{L}$ there cannot be conflicting choices. Note that, the game will deviate from the games used in the semantics of dDG$\mathcal{L}$ as our notion of hybrid automata allows for infinite traces, whereas hybrid programs only consider finite prefixes of these.

In order to turn an automaton into a game, we make some parts of the automaton controllable. That is, we partition the discrete transitions into two sets and give each player control over one of these.

Definition 49 (Hybrid Game Automaton)**.** *A* hybrid game automaton *is a structure* $HG = (S, E_c, U_c, l)$ *that consists of a hybrid automaton*

$$S = (U, X, L, E, F, Inv, Init) \ ,$$

a set of controllable transitions $E_c \subseteq E$*, a set of controllable input variables* $U_c \subseteq U$*, and a distinct location* $l \in L$*.*

Play. A game is *played* on a hybrid game automaton according to the following rules on the states of the hybrid automaton denoted by S. The possible moves of Demon are determined by the uncontrollable transitions $E \setminus E_c$ and the corresponding invariants and guards. Angel plays on the controllable transitions E_c. In addition, Angel always proposes a Lebesgue-measurable function that gives the future valuations of the variables in U_c until the next move is determined. Demon does so for the variables in $U \setminus U_c$. At every state of the game each player chooses an action that is either a finite number of discrete transitions (uncontrollable transitions for Demon and controllable ones for Angel) or a time period they want to let pass. The players are only allowed to let time pass if this is possible within the current evolution domain. Demon may also choose to take no action. In that case the action chosen by Angel is executed. If both players choose discrete transitions then Demon gets precedence and all transitions of Demon are executed. Subsequently, the transitions proposed by Angel

are executed, if they are still enabled. If both players choose to let time pass, the smaller amount of time is taken. In case one player chooses a discrete transition and the other one chooses to let time pass, the discrete transition gets precedence.

Winning. Demon *wins*, if he can force the game to enter the location l or if Angel does not have any more moves. Angel *wins*, if he can assert that the location l is avoided.

Observe that, the case where Angel has no more moves can happen if for example the system is on the edge of an invariant region and thus a discrete transition has to happen, but no transition is enabled.

Strategy. As *strategies* the players are allowed to use feedback control. That is, the decisions which actions to perform are based on the current state $(x, l) \in X \times L$. For a given game, a strategy is non-Zeno if there is no time t such that the runs of the automaton trigger the strategy to propose an infinite amount of actions within a time interval of length t.

Winning Strategy. We call a strategy a *winning strategy* for Demon (resp. Angel) if it is non-Zeno and for all non-Zeno strategies of Angel (resp. Demon) it is winning for Demon (resp. Angel).

5.1.2 Encoding our Robust Refinement Relation

To translate the question whether a system robustly refines another into such a hybrid game automaton, we encode the restrictions of the refinement relation into a hybrid automaton that is able to check whether either the distance between the system states is too large, or whether we are not able to find a suitable retiming at a certain point.

The locations of this automaton result from building the Cartesian product of the two systems we like to compare. We add additional copies of each location (named $\hat{l} \in \hat{L}$) that are used as targets of the transitions controlled by Demon in order to allow Angel to react to those actions. Furthermore, a distinct location *bad* is used to mark situations where either the spatial or the temporal distance are exceeding their bounds.

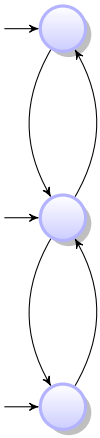
The retiming is modeled by speeding up/slowing down the system dynamics. This is done by a fresh variable s with values restricted to some bounded interval. We choose $s \in [0, 2]$. We now multiply the dynamics of

the first system with s and those of the second system with $2-s$. As s can be altered arbitrarily we can emulate all possible retiming relations. For example $s = 1$ means that both systems run with the same speed. Whereas $s = 0$ means only the second system is evolving and for the case $s = 2$ only the first one is. We keep track of the temporal distance of the systems in the variable r that represents the integral over $2s - 2$. That r indeed models the temporal distance can be seen if one considers the evolution of local clocks. A clock in the first system evolves with rate s while a clock in a second system evolves with $2 - s$. In case $s = 1$ both clocks evolve with the same speed of 1. Otherwise, we have a clock drift of $s-(2-s) = 2s-2$.

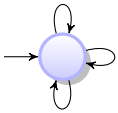
The spatial distance can be checked directly. We add invariants that force the automaton to go to the *bad* location if the distance is too large. Most of the controllable transitions are only enabled as long as the system variables and timings are close. This is not directly enforced in cases where Angel reacts on some discrete action chosen by Demon, i.e., an uncontrollable transition. Therefore, all uncontrollable transitions lead to a location (second component is in $\hat{L}$) where Angel might react. However, in these locations no time must pass which is enforced using the fresh clock c. We call these locations committed. The clock c is reset on all transitions that lead toward a committed location and the invariant of the location states the constraint $c \leq 0$. Hence it ensures that no positive evolution time is possible there that does not violate this evolution domain constraint. Recall that this means that neither player may choose to let time pass.

Example 18. *Let us sketch this idea in a picture. Consider the two abstract automata given in Figure 5.1. The specification has three locations and four transitions. The implementation consists of a single location with three transitions. In Figure 5.2 we sketch the refinement product of a these two systems. The unlabeled locations are the Cartesian product of the two automata. The locations labeled with C are copies used to allow Angel to react on the actions of Demon. Thus, all the transitions of the implementation (drawn with straight lines) lead to one of these locations. There are three of those transitions matching the three transition in the original implementation automaton. The outgoing transitions are those of the specification (drawn with dashed lines) and an additional one that models a no-op choice for Angel. Only in the original locations exist outgoing transitions towards the "bad" state.*

We now formalize this idea. First, let us define how to speed up or slow down a continuous evolution.



(a) Specification



(b) Implementation

Figure 5.1: Abstract examples for hybrid automata

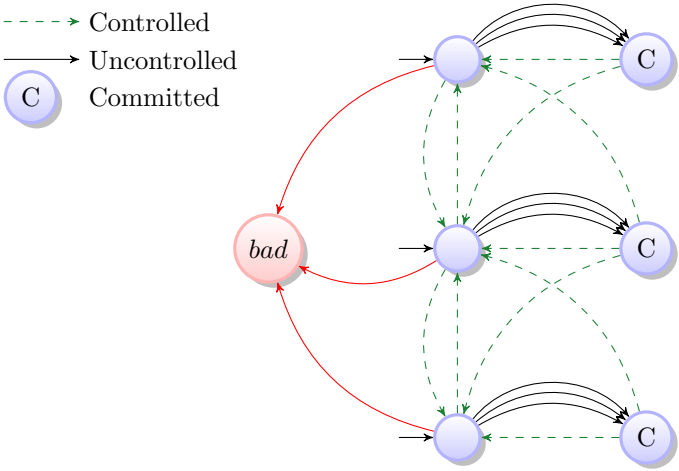


Figure 5.2: Relaxed refinement game sketch

Definition 50. *Let S be a system of differential equations*

$$\dot{x}_1 = f_1(x_1, \ldots, x_n), \ldots, \dot{x}_n = f_n(x_1, \ldots, x_n) \ .$$

We denote by $t \times S$ the system where the slope of each variable is multiplied with an expression t that does not contain the variables $x_1, \ldots, x_n$, i.e., the system

$$\dot{x}_1 = t \cdot f_1(x_1, \ldots, x_n), \ldots, \dot{x}_n = t \cdot f_n(x_1, \ldots, x_n) \ .$$

Let $\nu(t)$ denote the value of the expression t in the state ν. For the case where $0 \leq \nu(t) < 1$ the evolution is slowed down because the slope of each variable is closer to 0. For $1 < \nu(t)$ the systems evolves faster than it would be without this additional factor. Obviously, $\nu(t) = 1$ does not change the system speed in any way as it is the neutral element w.r.t. multiplication.

Definition 51 (Robust Refinement Product)**.** *Given two real numbers ε, δ, a hybrid automaton $A = (U_A, X_A, L_A, E_A, F_A, Inv_A, Init_A)$, and another hybrid automaton $B = (U_B, X_B, L_B, E_B, F_B, Inv_B, Init_B)$, we define the* robust refinement product*, w.l.o.g. assuming $(X_A \cap X_B) \cup (U_A \cap U_B) = \emptyset$ as the following hybrid game automaton $SG = (A \lessdot B, E_c, \{s\}, bad)$, where*

- $A \lessdot B = (U_\lessdot, X_\lessdot, L_\lessdot, E_\lessdot, F_\lessdot, Inv_\lessdot, Init_\lessdot)$
- *The variables of the resulting system are given by $U_\lessdot = U_A \cup U_B \cup \{s\}$ and $X_\lessdot = X_A \cup X_B \cup \{r, c\}$ where w.l.o.g. $(V_A \cup V_B) \cap \{s, r, c\} = \emptyset$.*
- *The locations are given by $L_\lessdot = (L_A \times (L_B \cup \hat{L}_B)) \ \dot\cup \ \{bad\}$, where $\hat{L}_B$ are duplicates of the original locations in L_B.*
- *Let χ be the following formula: $||x_A - x_B|| < \delta \wedge |r| < \varepsilon$, where x_A and x_B are the state vectors of the systems A and B respectively. Furthermore, let χ_s be the formula $||x_A - x_B|| \leq \delta \wedge |r| \leq \varepsilon$.*
- *The set of discrete transitions $E_\lessdot$ is the smallest set such that:*
  - *If $(l_A, \phi, l'_A) \in E_A$ then for all $l_B \in L_B$,*

    $$((l_A, l_B), \phi \wedge \chi \wedge c' = 0, (l'_A, \hat{l}_B)) \in E_\lessdot \ .$$

  - *If $(l_B, \phi, l'_B) \in E_B$ then for all $l_A \in L_A$,*

    $$((l_A, l_B), \phi \wedge \chi, (l_A, l'_B)) \in (E_\lessdot \cap E_c)$$

    and

    $$((l_A, \hat{l}_B)), \phi, (l_A, l'_B)) \in (E_\lessdot \cap E_c) \ .$$

  - *For all* $l_A \in L_A$ *and all* $l_B \in L_B$, $((l_A, l_B), \neg\chi, bad) \in E_{\lessdot}$.
  - *For all* $l_A \in L_A$ *and all* $l_B \in L_B$,

$$((l_A, \hat{l}_B), true, (l_A, l_B)) \in (E_{\lessdot} \cap E_c) \ .$$

- *For all* $l = (l_A, l_B) \in L_{\lessdot}$ *or* $l = (l_A, \hat{l}_B) \in L_{\lessdot}$ *we construct* $F_{\lessdot}(l)$ *as*

$$\dot{r} = 2s - 2 \wedge \dot{c} = 1 \wedge s \times F_A(l_A) \wedge (2 - s) \times F_B(l_B) \ .$$

- *The invariants of the locations are given by* $Inv_{\lessdot}$ *which assigns each location* (l_A, l_B) *or* $(l_A, \hat{l}_B)$ *an invariant of the form*

$$Inv_A(l_A) \wedge Inv_B(l_B) \wedge \chi_s \wedge 0 \leq s \leq 2 \ .$$

  If $l = (l_A, \hat{l}_B)$ *we further add* $c \leq 0$.

- $Init_{\lessdot} = \{((l_A, l_B), Init_A(l_A) \wedge Init_B(l_B)) \mid l_A \in L_A \wedge l_B \in L_B\}$

Using this game, we can determine whether two systems stand in refinement relation as defined in Definition 22 (see page 44).

Theorem 7. *Given two hybrid automata* A *and* B. *If there is a winning strategy for Angel in the game played on the hybrid game automaton* $(A \lessdot B, E_c, \{s\}, bad)$ *then* $A \wr\xrightarrow{\varepsilon,\delta}\ B$ *holds.*

Proof. Assume that there is a winning strategy for Angel but $A \wr\xrightarrow{\varepsilon,\delta}\ B$ does not hold. From the latter, we know that there is a run of system A such that, no matter which retiming is applied, system B cannot stay close enough. Let this run be σ. We now construct a winning strategy for Demon using σ. Demon chooses his actions in a way that the valuations of the variables of the first system at time t coincide with $\sigma(t - r)$. Angel is not able to influence the valuations of the variables at those points, as the discrete transitions that it can choose from are those of B. He is also not able to restrict the movement of Demon, as the intermediate locations only allow him to react on actions performed by Demon and as he looses if there is a time deadlock, he can also not avoid time going to infinity. This, as there was no run of B that stays close enough to σ, eventually leads to a state where the condition $||x_A - x_B|| < \delta \wedge |r| < \varepsilon$ is violated. In this state Demon can choose to enter the location *bad*. This contradicts the assumption that there is a winning strategy for Angel and thus concludes the proof. □

Note that the reverse implication does not hold, as the game demands a tighter coupling of the system behaviors with regard to branching than our notion of ε-δ-refinement. If the systems have continuous inputs, this becomes even more clear as all those inputs are controlled by Demon and can therefore be used to drive the system towards *bad* instead of being identical for both systems at the same points in time.

Corollary 7. *If we restrict the possible moves of Angel by adding differential equations describing the evolution of s and she is still able to win the game, then the systems are in refinement relation.*

This follows directly from the fact that Theorem 7 demands the existence of a winning strategy. Now, if we can find a winning strategy for a system where the control of s is further restricted, we still can assert that there is a winning strategy in the original game.

A good candidate for a strategy for controlling s is optimal control (see e.g. [PBGM62]) with the goal of minimizing the value of $||x_A - x_B||$. Here, optimal control refers to the idea of synthesizing a continuous controller that minimizes the distances between the two trajectories.

Example 19. *For example, let us consider the case of $||\cdot|| = ||\cdot||_e$ being the Euclidean norm. As the Euclidean norm contains a square root we w.l.o.g. take the square of the distance as minimization target. This yields equivalent results as the square root is a monotone transformation. For $x_A = (x_{A,1}, \ldots, x_{A,n})$ and $x_B = (x_{B,1}, \ldots, x_{B,n})$, the square of the distance evolves as follows:*

$$\begin{aligned}\frac{d(||x_A - x_B||_e)^2}{dt} &= \frac{d(\sqrt{((x_{A,1} - x_{B,1})^2 + \cdots + (x_{A,n} - x_{B,n})^2)}^2)}{dt} \\ &= \frac{d((x_{A,1} - x_{B,1})^2 + \cdots + (x_{A,n} - x_{B,n})^2)}{dt} \\ &= \Sigma_{i=1}^{n}(2(x_{A,i} - x_{B,i}) \cdot (s\frac{dx_{A,i}}{dt} - (2-s)\frac{dx_{B,i}}{dt}))\end{aligned}$$

Let s_{min} be the s that minimizes this term. Now choose the input s in the following way: If $r < \varepsilon \wedge s_{min} > 1$ or $r > -\varepsilon \wedge s_{min} < 1$ choose $s = s_{min}$. Otherwise choose $s = 1$. The resulting strategy for controlling s can then be encoded into a hybrid automaton and included into the original automaton.

The choice of the strategy is motivated by the fact that the location *bad* can only be entered if the distances between the two systems become too large. As we might be able to trade spatial distance against temporal distance we choose to minimize the spatial distance. However, this strategy is only an heuristic as there are systems, where it is necessary to let the spatial distance increase a bit to for example unify switching timings, as the guard effects might otherwise lead to a violation of the bounds on the spatial distance (see Example 5 on page 37).

Definition 52. *A hybrid automaton is considered* deterministic *if (1) all of its transitions are urgent, i.e., all the guards of the transitions are overlapping in a singular point with the border of its sources invariant and all trajectories of the mode are pointing outwards of the invariant region at that point, and (2) for each point in time, at most one transition is enabled.*

Remark 6. *Let A be a hybrid automaton and B be a deterministic hybrid automaton. If we modify $A \lessdot B$ in a way that the assumptions of Corollary 7 are satisfied, assume that the system values are identical at the initial locations, and are able to show that on all runs of this modified version of $A \lessdot B$ the location bad is avoided, then, by Corollary 7, $A \rightsquigarrow^{\varepsilon,\delta} B$ holds. The assumptions of Corollary 7 could, e.g., be satisfied by using the optimal control strategy. Thus, we can use a model checker (e.g. FOMC [DDD+12], PHAVer [Fre08], SpaceEx [FGD+11], or HSolver [RS07], depending on the system class and complexity) to search for a certificate for the fact that the location bad is unreachable.*

Remark 7. *On a similar line of thought, if one can prove that both systems stabilize within a certain time it also possible to use bounded model checking up until a stable state has been reached. This could be performed, e.g., using iSAT [FHT+07].*

5.2 dDG$\mathcal{L}$ and Similarity

As the approach in the previous section was limited by the assumption that specifications are deterministic and the available tools relying on concrete numbers instead of abstract parameters as free function symbols, we now present an approach for similarity of parametric hybrid systems based on our game logic dDG$\mathcal{L}$. That is, in this section we discuss how to show that

a hybrid system α modeled as a hybrid program ε-δ-refines by another hybrid system β also modeled as a hybrid program.

The idea is, like in the previous section, to run α and β in parallel and give Falsifier control over the actions of α and thus constructing some trace of that system. Verifier on the other hand gets control over the decisions that are possible for system β. The winning condition for Verifier is that the distance between the variables α and those of β is at most δ. If there is a trace $\sigma_\alpha \in \tau(\alpha)$ of α for which there is no trace $\sigma_\beta \in \tau(\beta)$ of β such that $\sigma_\alpha \xleftarrow{\varepsilon} \wr \xrightarrow{\delta} \sigma_\beta$ then Falsifier can take the actions to produce this trace and Verifier will have no way of ensuring that the variables of β will stay within limits and thus lose the play. Therefore, if there is a winning strategy for Verifier then there is no such trace of α and α is an ε-δ-refinement of β.

Again, like in the previous section, we add a speedup factor s that is controlled by Verifier to model the retiming and measure the temporal distance by a variable r. Verifier is responsible of ensuring that $|r|$ never exceeds ε.

5.2.1 Trace Equivalence

To relate traces of hybrid programs as well as plays of hybrid games, we define an equivalence relation with respect to their common variables.

Let $Var(\sigma)$ denote all variables occurring in the hybrid trace σ. Furthermore, let $\pi_X(\sigma)$ be the projection of a trace σ to the variables in the set X.

As the semantics keeps track of every discrete action, even if it has no effect on the variable valuations we define a compression function that enables us to consider traces as equivalent even though they perform actions that do not change the variable valuations in between.

Definition 53 (Trace Compression)**.** *For a trace $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$, we denote by $compress(\sigma)$ the exhaustive application of the following operation: If there is an i such that $\sigma_i(max(\mathrm{dom}\ \sigma_i)) = \sigma_{i+1}(0)$ then replace σ_i, σ_{i+1} by*

$$\sigma'(t) = \begin{cases} \sigma_i(t) & \text{if } t < max(\mathrm{dom}\ \sigma_i) \\ \sigma_{i+1}(t - \max(\mathrm{dom}\ \sigma_i)) & \text{otherwise} \end{cases}.$$

Using this function we can now define when we consider traces equivalent. That is if they are identical after first projecting them to the common variables and then applying the compression function.

Definition 54 (Trace Equivalence)**.** *We say that two traces σ_1 and σ_2 are* equivalent *with respect to their common variables, denoted by $\sigma_1 \equiv \sigma_2$ iff*

$$compress(\pi_{Var(\sigma_1)\cap Var(\sigma_2)}(\sigma_1)) = compress(\pi_{Var(\sigma_1)\cap Var(\sigma_2)}(\sigma_2)) \ .$$

Let us illustrate this equivalence relation.

Example 20. *Consider the following traces for a system with a single variable v that are all equivalent under compression:*

- $\sigma_1 = (\{(t, \{v \mapsto x\}) \mid x = 2t \wedge 0 \leq t \leq 10\})$
- $\sigma_2 = (\sigma_{2,1}, \sigma_{2,2})$ *where* $\sigma_{2,1} = \{(t, \{v \mapsto x\}) \mid x = 2t \wedge 0 \leq t \leq 4\}$ *and* $\sigma_{2,2} = \{(t, \{v \mapsto x\}) \mid x = 2t + 4 \wedge 0 \leq t \leq 6\})$
- $\sigma_3 = (\sigma_{3,1}, \sigma_{3,2}, \sigma_{3,3})$ *where*

$$\begin{aligned}
&\sigma_{3,1} = \{(t, \{v \mapsto x\}) \mid x = 2t \wedge 0 \leq t \leq 4\},\\
&\sigma_{3,2} = \{(0, \{v \mapsto 8\})\}, \textit{ and}\\
&\sigma_{3,3} = \{(t, \{v \mapsto x\}) \mid x = 2t + 4 \wedge 0 \leq t \leq 6\}) \ .
\end{aligned}$$

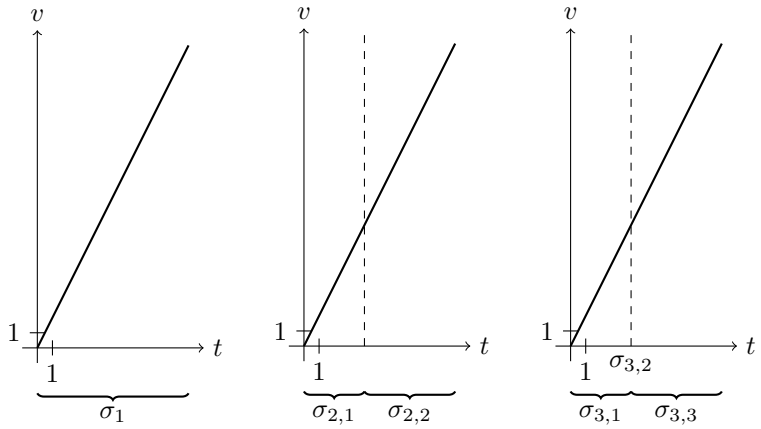
All these traces look the same as illustrated in Figure 5.3. Here, the time axis reflects the axis after composition. Otherwise, the dashed line would mark the begin of a new axis labeling. The first two traces could for example be produced by the following simple system

$$(\dot{v} = 2)^* \ .$$

A system that can only produce the first trace is $\dot{v} = 2$. The third system could also be produced by a system that features an additional discrete action that does not change the variable valuations, for instance a test. Thus, the system

$$(\dot{v} = 2; ?true)^*$$

is able to produce the third, but not the first two traces. Observe that, under compression all these traces collapse to the first one and are thus considered equivalent.



(a) Sketch of σ_1 (b) Sketch of σ_2 (c) Sketch of σ_3

Figure 5.3: Equivalent traces over a unified time axis t

5.2.2 Standard Form

In order to get a good handle on hybrid programs for this constructions we define a standard form.

Definition 55 (Standard Form)**.** *A hybrid program α is in* standard form *if it has the following structure:*

$$\alpha \hat{=} \left(v_\alpha := *; ?init; \left(\bigcup_{i \in I} (D_i; (C_i \wedge H_i)) \right)^* \right)$$

where $I \subset \mathbb{N}$ is finite, D_i are discrete hybrid programs, C_i are continuous evolutions, H_i is the evolution domain constraint, and init describes the initial region. The variables occurring in α are denoted by v_α and $v_\alpha := *$ assigns values to all of these. In the following we will write $D_i(\alpha)$ to denote D_i for a program α that is in standard form. Likewise, we will use $C_i(\alpha)$, $H_i(\alpha)$, and $init(\alpha)$.

Observe that the D_i are regular programs over real valued variables. They might contain loops, however they must not contain any continuous

evolutions. Contrary to that the C_i are continuous evolutions that must not contain any discrete assignments.

Not every hybrid program can be transformed into this standard form and still produce the same traces. One such example is the following:

$$y := 0; \dot{y} = 1; ?y > 5$$

However, this program mixes system properties and system behavior. The traces produced by this system have a minimum length of 5. However, for every system in the physical world we can continuously observe its behavior. Therefore, arbitrary short runs should exist. Of course, there are cases where the property we want to investigate only makes sense for runs that have a certain minimum length. In those cases the property descriptions should be adapted accordingly instead of altering the system description. Therefore, we restrict ourselves to those programs that can be formulated as hybrid automata. In order to show that our standard form is not too restrictive, we show that it covers all hybrid automata.

Theorem 8. *Every hybrid automaton can be transformed into a hybrid program in standard form such that their finite traces are equivalent w.r.t. to the variables occurring in the original automaton.*

We can prove this similarly to the relation of hybrid automata and hybrid programs proven in [Pla10b] w.r.t. the reachable states.

Proof of Theorem 8. For a hybrid automaton $A = (U, X, L, E, F, Inv, Init)$ we construct a hybrid program the following way. First, we fix a distinct variable ℓ to keep track of the location. Assume w.l.o.g. that the same variable was used in the automaton semantics to store the location. Now we build a program $p(A)$ as sketched in Figure 5.4 where we use vectorial assignments for $x = (x_1, \ldots, x_n)$ the variables in X and existential quantification over all variables in U expressed by the quantifier $\exists u$. The $x' = (x'_1, \ldots, x'_n)$ is used to denote the post values while taking a discrete transition, like in the semantics of the hybrid automaton.

It is easy to see that the constructed program is in standard form. We now show that the traces produced by this program coincide w.r.t. to the variables in X with the finite ones of the original automaton.

We start with the direction that all traces of the hybrid automaton can be simulated by the hybrid program. We do this by induction over the number n of transitions (discrete or continuous) taken.

$$
\begin{array}{l}
x := *;\ \ell := *; \\
?\left(\bigwedge_{\ell_i \in L} (\ell = \ell_i \to Init(\ell))\right); \\
\Big(?\ell = \ell_i; \exists u\ F(\ell_i) \wedge Inv(\ell_i) \\
\quad \cup \bigcup_{(\ell_i, g, \ell_j) \in E} (?\ell = \ell_i; x' := *; ?g; x := x'; \ell := \ell_j; \exists u\ F(\ell_j) \wedge Inv(\ell_j)) \\
\quad \cup \ldots \\
\Big)^*
\end{array}
$$

Figure 5.4: Program $p(A)$ encoding some hybrid automaton A

IA If $n = 0$ then the trace produced by the hybrid automaton is just σ_0. The program started in $first(\sigma_0)$ produces the trace $\sigma_0, \sigma_1, \sigma_2$ using zero iterations of the loop. However, the compression reduces this to σ_0 as no state changes happen.

IH Assume that for all $0 \leq i \leq n$ for all traces of length i of A there is a trace of $p(A)$ that is equivalent.

IS We consider the trace $\sigma_0, \ldots, \sigma_{n+1}$. By induction hypothesis, we have that for the prefix $\sigma_0, \ldots, \sigma_n$ there already is an equivalent trace of the hybrid program $p(A)$.

- Consider the case where the last step of the automaton was a discrete transition e. We then choose the branch created from e during an additional execution of the loop. Then we choose an evolution time of 0 for the continuous evolution. Thus, under compression we get a trace that is equivalent to $\sigma_0, \ldots, \sigma_{n+1}$.
- In case the last step was a continuous evolution, then we choose to stay in the same location, therefore choosing the branch with the correct location that was not created by some edge. Using the same evolution time, we can create a run that under compression is equivalent to the original one.

Now we turn our focus to the opposite direction. That is, we prove that every run of the hybrid program is equivalent to one of the original automaton. We do so by induction over the number of loop iterations n.

IA For $n = 0$ the produced run is of the form $\sigma_0, \sigma_1, \sigma_2$. However under compression it becomes just σ_0 and is thus equivalent to the run taking no transition of the automaton.

IH For all $0 \leq i \leq n$ for the traces produced within i loop iterations, there is a trace produced by the automaton that is equivalent.

IS Consider a run of length $n + 1$. In case the branch taken during the $n + 1$-st execution of the loop was constructed from an edge, then this edge is enabled after matching the trace up until that point. Thus by taking this edge during the execution of the automaton and then letting time pass for the same amount of time as in the execution of the program it produces a run that is equivalent to that of the program. In case the branch taken was not constructed from an edge then the automaton just needs to let the same amount of time pass in the current location. □

5.2.3 Encoding Similarity in dDG$\mathcal{L}$

We have provided a suitable standard form and proved that it is not too restrictive. Given two hybrid programs in standard form, we now construct a game (see Figure 5.5) such that if Verifier has a winning strategy then one system is a ε-δ-refinement of the other. First, we add some auxiliary variables s, r for controlling and measuring the retiming, t, t_β as clocks for measuring cycle and minimal dwell times, and p_α, p_β as program counters. Furthermore, we add c_β to denote the minimal dwell time of β, and t_s to denote the sampling time of the retiming function. This sampling time is also used to ensure that Verifier is able to execute discrete actions and change modes at least every t_s time units.

To ensure that the distance between the valuations of the system variables stays small enough, we run the systems in parallel. Using the standard form, we can construct the Cartesian product easily, by first evaluating the mode choice of α (lines 5 and 6) and subsequently of β (lines 7 and 8). As we have to make sure that Verifier chooses branches that allow actual progress in the system evolution, we test that time can pass in the current mode (line 11). Then, we run the continuous evolutions of the two systems concurrently (lines 12-14). Like in the previous section, to allow for

retiming, the dynamics of α are multiplied by s and those of β by $2 - s$. For $s \in [0, 2]$ this allows for arbitrary evolution speed differences. Here, s can be updated at least every t_s time units. We use r to measure the retiming. Its derivative is $2s-2$ which is exactly the rate with which clocks of the two systems diverge/converge. The notion of ε-δ-refinement forces an upper bound on this retiming. This is ensured by running the two systems at the original speed, once the distance, i.e., $|r|$ becomes too large (line 13). After every continuous evolution Verifier has to show that the spatial distance between the variable values of the two systems is within the specified bounds (line 15).

To ensure that a strategy exists to mirror the original behavior of the system α, it is necessary to allow "no operation" actions at two points. First, the continuous evolution might be stopped due to hitting an evolution domain constraint of β or because the sampling time is elapsed. It might be the case that at these time points no discrete action of the original system α was performed. To simulate this behavior, no discrete action is necessary, if no mode change is done and the continuous evolution can still continue for some positive amount of time without being restricted by the evolution domain constraint (line 5). The same holds for the discrete part of β (line 7). In addition Falsifier might choose to not take any action at all during the loop execution (line 18). This might become necessary, as we construct overapproximations of the necessary loop iterations. However, this number has to be provided to Verifier in advance and, thus, he might force faster progress in terms of the number of loop iterations than what is necessary if he does not cooperate. To avoid issues here, we allow for stuttering off these additional loop iterations using the $[?true]$ choice.

Definition 56 (Robust Refinement Game)**.** *For two hybrid programs α, β in standard form with disjoint variable sets, i.e., $V(\alpha) \cap V(\beta) = \emptyset$, and index sets such that $0 \notin (I_\alpha \cup I_\beta)$, and*

$$\{r, s, t, t_\beta, p_\alpha, p_\beta, c_\beta, t_s\} \cap (V(\alpha) \cup V(\beta)) = \emptyset \ ,$$

we define a game $\mathcal{G}_{\varepsilon,\delta}(\alpha, \beta)$ *as shown in Figure 5.5 where*

$$\Psi(s, r) \mathrel{\hat{=}} (r = -\varepsilon \wedge s > 1) \vee (r = \varepsilon \wedge s < 1) \ ,$$

and $a \times C$ is defined like in Definition 50 (see page 129).

The branch variables p_α and p_β are necessary in order to avoid changing the mode without executing the discrete controller part for this mode.

$$
\begin{aligned}
&[s := 1; r := 0; t := 0; t_\beta := 0; p_\alpha := 0; p_\beta := 0] \\
&[v_\alpha := *; ?init(\alpha)] \\
&\langle c_\beta := *; ?c_\beta > 0; t_s := *; ?t_s > 0\rangle \\
&\langle v_\beta := *; ?init(\beta)\rangle \\
&\Bigg(\Bigg(\bigcap_{i \in I_\alpha} \Big(\big(\big[?|p_\alpha| = j \wedge \langle t := 0; (C_i(\alpha) \wedge H_i(\alpha), \dot{t} = 1)\rangle t > 0; p_\alpha := -i \big] \\
&\qquad \cap [p_\alpha := i; D_i(\alpha)] \Big) \\
&\qquad \bigcup_{j \in I_\beta} \Big(\big(\langle ?|p_\beta| = j \wedge \langle t := 0; (C_j(\beta) \wedge H_j(\beta), \dot{t} = 1)\rangle t > 0; p_\beta := -j \big\rangle \\
&\qquad\quad \cup \langle p_\beta := j; D_j(\beta); t_\beta := 0\rangle \big) \\
&\qquad\quad \langle s := *; ?0 \leq s \leq 2\rangle \\
&\qquad\quad \langle ?t < t_s \cup t := 0\rangle \\
&\qquad\quad \langle ?t_\beta < c_\beta \rightarrow \langle C_j(\beta) \wedge H_j(\beta), \dot{t}_\beta = 1\rangle t_\beta \geq c_\beta\rangle \\
&\qquad\quad [\ (\ (\neg\Psi(s,r) \wedge s \times C_i(\alpha) \wedge (2-s) \times C_j(\beta) \wedge \dot{r} = 2s - 2) \\
&\qquad\qquad \vee (\Psi(s,r) \wedge C_i(\alpha) \wedge C_j(\beta)) \\
&\qquad\quad), \dot{t}_\beta = 1, \dot{t} = 1, H_i(\alpha) \wedge H_j(\beta) \wedge t \leq t_s \big] \\
&\qquad\quad \langle ?||v_\alpha - v_\beta|| \leq \delta \wedge |r| \leq \varepsilon\rangle \\
&\qquad \Big) \\
&\quad \Big)\Big) \\
&\quad \cap [?true] \\
&\Bigg)^{[*]}
\end{aligned}
$$

Figure 5.5: Construction of $\mathcal{G}_{\varepsilon,\delta}(\alpha, \beta)$

Note that it is initially 0. As $0 \notin (I_\alpha \cup I_\beta)$ we are sure that during the first execution of the loop, a discrete action is executed. Observe that, the continuous evolution in lines 12-14 is not in the strict sense part of our syntax. However it can be interpreted as a differential algebraic constraint [Pla10b] and dealt with accordingly. That is, we need to create a disjunction normal form and split along the disjunctions. We then replace the differential equation system by a loop with a nondeterministic choice over the resulting alternatives.

Furthermore, this encoding includes programs in which tests occur that feature $\mathsf{d}\mathcal{L}$ formulas instead of first-order formulas over the reals. This is a straight forward extension to the syntax and semantics of hybrid programs called *rich tests* [Pla10b]. We keep the semantics of instantaneous tests and use the semantics of $\mathsf{d}\mathcal{L}$ formulas to test whether the formula is satisfied in the current state instead of that of first-order logic over the reals. This gives a well-defined semantics for $\mathsf{dDG}\mathcal{L}$ formulas. Note that the test still does not change the state. For example the following formula:

$$t = 0 \rightarrow \left[?\langle \dot{t} = 1 \rangle t > 0\right] t = 0 \tag{5.1}$$

For an initial state where $t = 0$ holds the program $?\langle \dot{t} = 1 \rangle t > 0$ terminates without changing the state. The test condition is satisfied as from this state there is a state reachable along the differential equation $\dot{t} = 1$ such that $t > 0$. However, the program does not move to this state. In case $t \neq 0$, the implication is trivially satisfied. Thus, the formula (5.1) is valid in $\mathsf{dDG}\mathcal{L}$.

As some specifications of hybrid systems lack real-world realizability and we want to bridge the gap between specifications and implementations, we do neither consider traces of a system that are Zeno, i.e., an infinite number of transitions is taken within a finite amount of time, nor behaviors that are time blocking. Time-blocking means that at a certain point in time, no future evolution of the system is possible due to an invariant preventing any continuous evolution and no action is possible to change this situation. For the system β we exclude these traces by adding the minimum dwell time requirement in line 11. Zeno runs of system α do not pose any issue for the approach presented here as they are not useful choices for Falsifier anyway.

Example 21. *The following example illustrates why we need the no-op choice in line 18. If β would have the following structure*

$$e := 0; ((?e = 0; e := 1; f := 1) \cup f := 0); t := 0; \dot{t} = 1 \wedge t \cdot f \leq 0)^*$$

it could stop the time exactly once. This is, if the program chooses the path where $f := 1$ is executed then the evolution domain constraint restricts the evolution to 0 time units. However, it is not forced to do so. Alternatively, it can always choose to execute the branch where $f := 0$ and, thus, the evolution domain constraint is a tautology, i.e. $t \cdot f = t \cdot 0 \leq 0$. Whenever, such effects are possible but not necessary, Falsifier cannot a priori exactly determine how many iterations it will take to simulate a given run of system α. In order to allow for overapproximation of this number, we have to allow for no-op executions of the loop. This way, in case Verifier does decide not to take such a blocking run, Falsifier can just stutter off the rest of the loop iterations.

We add the winning condition

$$||v_\alpha - v_\beta|| \leq \delta \wedge |r| \leq \varepsilon$$

for Verifier in order to ensure that in every winning state our ε-δ-refinement relation between α and β holds. Further, we now show that if there is a winning strategy for Verifier, i.e., $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)(\delta \wedge |r| \leq \varepsilon)$ is valid, the hybrid system α ε-δ-refines the hybrid system β, i.e., $\alpha \wr\xrightarrow{\varepsilon,\delta}\beta$ holds.

We now have to show that $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ indeed witnesses that $\alpha \wr\xrightarrow{\varepsilon,\delta}\beta$ holds. For this, we proceed along the following path. We first show that for every trace σ_α of α there is a strategy for Falsifier that no matter what Verifier does the trace σ_p produced by $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ is equivalent to that σ_α, i.e., $\sigma_\alpha \equiv \sigma_p$. Then we show that for all strategies of Falsifier and Verifier the trace σ_p produced by $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ is equivalent to some trace σ_β of β, i.e. $\sigma_p \equiv \sigma_\beta$. Together with the fact that there can only be a winning strategy for Verifier if the spatial and temporal bounds are respected we can conclude that $\alpha \wr\xrightarrow{\varepsilon,\delta}\beta$ holds. This agenda is sketched in Figure 5.6.

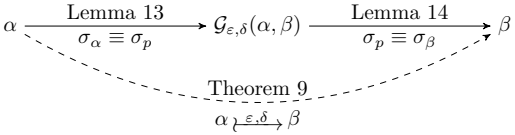


Figure 5.6: Sketch of the proof of Theorem 9

To relate traces of the original systems α, β, and the constructed game $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ we have to account for the effect of the retiming to the underlying

semantics. Observe that, therefore the illustration in Figure 5.6 was oversimplifying the process as the traces are not equivalent but only equivalent after reverting the effect of the retiming. Therefore, we define a relation that reverts its effect. The idea behind the reverse retiming is as follows: The variable s was used in our game construction to speed up or slow down the evolution of a specific system part. As it was changed in a discrete way, we get separate trace parts for each of those. In each of these parts, we can reconstruct the original speed by multiplying the time axis by the multiplicative inverse of what it was stretched with. In case the evolution was stopped completely, no change to the variables happened. We can thus compress this to a point trace. However, once the distance between the two system grew too large they run with the original speed. Therefore, no transformation with respect to s is necessary then. Still, we have to shift the time points in order to reconstruct the original time axis labels.

Definition 57 (Reverse Retiming)**.** *For a trace $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$, a term θ, and a formula ϕ, we define the* reverse retiming $ret^{-1}(\sigma, V, \theta, \phi)$ *as the sequence of relations*

$$(ret^{-1}(\sigma_0, V, \theta, \phi), ret^{-1}(\sigma_1, V, \theta, \phi), ret^{-1}(\sigma_2, V, \theta, \phi), \dots)$$

where

$$\begin{aligned} ret^{-1}(\sigma_i, V, \theta, \phi) = &\Big(\left\{ \left(\frac{t}{\nu(\theta)}, \pi_V(\nu) \right) \mid (t, \nu) \in \sigma_i \wedge \nu(\theta) \neq 0 \wedge \nu \not\models \phi \right\} \\ &\cup \{(0, \pi_V(\nu)) \mid (t, \nu) \in \sigma_i \wedge \nu(\theta) = 0 \wedge \nu \not\models \phi\}\Big) \\ &\circ \{(t - \min(\operatorname{dom} \varsigma_i), \pi_V(\nu)) \mid (t, \nu) \in \varsigma_i\} \end{aligned}$$

where

$$\varsigma_i = \{(t, \nu) \in \sigma_i \mid \nu \models \phi\}$$

and V is a set of variables such that the trace composition is defined.

Note that the reverse retiming yields a function if for all states ν it holds that $\nu(\theta) \neq 0$. Further note that for the traces generated by $\mathcal{G}_{\varepsilon,\delta}(\cdot,\cdot)$ the value of s is constant for each σ_i as it is only updated discretely. Therefore, if we choose θ only dependent on s, its value is also constant for each σ_i. This means that either $[0, 0]$ maps to all values or the domain is transformed in an order-preserving way. The formula ϕ, provided as a parameter to the reverse retiming, is used to determine whether the control choice for the

θ had an effect or not. In our construction of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ the continuous evolutions evolve with normal speed implying the ignorance of the choice of s in case the maximal value of r is reached and the system is about to violate its temporal bounds.

Now we come to the first important step in order to prove a refinement relation between α and β. The following lemma links the traces of α to those of our robust refinement game $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$. We choose an arbitrary trace σ_α of system α and show that if there is winning strategy for Verifier then there is a strategy for Falsifier such that, whatever Verifier does that is allowing him to win, the variables of system α follow a trace σ_p in the execution of the game that is equivalent up to retiming to σ_α.

Lemma 13. *If there is a winning strategy for Verifier in $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$, then for every trace $\sigma_\alpha \in \tau(\alpha)$ with $last(\sigma) \neq \Lambda$ there is a Falsifier strategy f such that for every Verifier strategy v that is winning from $first(\sigma_\alpha)$ in the game $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ the resulting trace σ_p where*

$$(\surd, \sigma_p) = \mathfrak{p}_{f,v}(\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)@first(\sigma_\alpha))$$

is equivalent with respect to the common variables up to retiming, i.e.,

$$\sigma_\alpha \equiv ret^{-1}(\sigma_p, V(\alpha), s, s < 1 \wedge r = \varepsilon)$$

where s arises in the construction of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ as given in Definition 56.

To make the proof easier, we add the program counters p_α and p_β to the programs to keep track of the decisions made by the players in the discrete part of the programs.

Definition 58 (Annotated Standard Form)**.** *We amend the definition of our standard form with branch labels. This gives the following* modified standard form*:*

$$\mathfrak{A}(\alpha) \hat{=} \left(v_\alpha := *; ?init; \left(\bigcup_{i \in I} (p_\alpha := i; D_i; (C_i \wedge H_i)) \right)^* \right)$$

where $I \subset \mathbb{N}$ is finite, D_i are discrete hybrid programs, C_i are continuous evolutions, H_i is the evolution domain constraint, and init describes the initial region. The variables occurring in α are denoted by v_α and $v_\alpha := *$ assigns values to all of these. In the following we will write $D_i(\alpha)$ to denote D_i for a program α that is in annotated standard form. Likewise, we will use $C_i(\alpha)$, $H_i(\alpha)$, and $init(\alpha)$.

In the proof of Lemma 13 we need to first show that for a given trace σ_α of α we can find a number of loop iterations for our game such that the resulting trace will mimic σ_α. In order to do so, we can use that whenever a mode is entered by β it can stay in that mode for at least some time. Should we guess a number of loop iterations that is too high, it can be stuttered off using the $[?true]$ choice. Furthermore, the variables of α are never altered directly by actions controlled by Verifier. In addition, we can use that the program is in annotated standard form. Hence the trace explicitly documents all discrete choices.

Proof of Lemma 13. Let $\sigma \in \tau(\mathfrak{A}(\alpha))$ be arbitrary. The trace σ has the form

$$\sigma_0, \cdots, \sigma_i, \sigma_{i+1}, \sigma_{k,l}, \cdots, \sigma_{m,l}, \sigma_{m+1,l}\sigma_{k,l+1}, \cdots, \sigma_{m,l+1}, \sigma_{m+1,l+1}, \cdots \ .$$

Here, 0 to i correspond to the initialization and $i+1$ is the test for the initial condition. The index l counts the loop iterations, where in each iteration indexes k to m give the discrete computations and $m+1$ the continuous evolution.

We now construct a strategy f for Falsifier in the game $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ such that, for every strategy v with which Verifier wins the game, the equivalence

$$\pi_{V_\alpha}(\sigma) \equiv ret^{-1}(\sigma_p, V(\alpha), s, r = \varepsilon \wedge s < 1)$$

holds where $(\surd, \sigma_p) = \mathfrak{p}_{f,v}(\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)@first(\sigma), \surd)$.

This strategy has to resolve the nondeterminism of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ for Falsifier. That means:

- In line 2, it has to choose initial valuations for the variables of α.
- In line 5 (to 19), it has to choose a number of loop iterations in the game.
- During every loop iteration, in line 5 and 6, it has to choose which branch of α to take and how to resolve the nondeterminism within the discrete part of this branch. Alternatively, it is allowed to choose to take no action ($[?true]$) at all (line 18).
- If a branch was chosen, it has to choose the evolution times for the continuous evolutions (in lines 12-14).

Note that line 1 also contains an action controlled by Falsifier. However, all the assignments are deterministic and, thus, there is only one choice for a compatible strategy.

Strategies of Verifier that are winning cannot delay the evolution in time arbitrarily long and can only produce a bounded number of interrupts during the continuous evolutions of α. This follows from the fact that β has to be able to stay in a mode for at least some time which is guaranteed by the test (in line 11)

$$?(t_\beta < c_\beta \to \langle C_j(\beta) \wedge H_j(\beta), \dot{t}_\beta = 1\rangle t_\beta \geq c_\beta) \ .$$

The clock t_β is reset every time Verifier performs a discrete action from the original program β in line 8. That way, it is ensured that he does not take any actions that would lead to convergence of the evolution times (i.e., a Zeno run). Therefore, there is a strategy of Falsifier s.t. the resulting trace matches. Using the auxiliary variables to determine which branch was chosen to generate σ, we construct this strategy as follows:

To match $\sigma_0, \cdots, \sigma_i, \sigma_{i+1}$ we choose the same initial valuations. The action controlled by Verifier in this initialization phase will later be removed by the projection to the variables of α anyway and, thus, disregarded by our notion of trace equivalence. The same holds for the variables that are used to control the retiming, and measure sampling as well as minimal dwell times.

The next important decision for Falsifier is the choice of the number of loop iterations. Therefore, we compute the number of loop iterations necessary in order to match σ for all possible initial values of the variables of β as well as the variables c_β and t_s:

1. Calculate the total number of loop iterations used to construct σ and compute the overall time the trace took. This gives the minimal number of iterations needed to match the trace.

2. Calculate the maximum number of interrupts caused by β based on the value of c_β and the length of the current trace. Here, we can use that the maximal trace in terms of variables in β is at most ε time units longer than the trace σ.

3. Calculate the maximum number of interrupts caused by the sampling of the retiming function.

4. Add these numbers to receive a sufficient amount of loop iterations n.

We choose n as the number of loop iterations and, subsequently, choose for first execution of the loop the same discrete branch as was taken in the construction of σ. For $D_i(\alpha)$, we then choose exactly the same trace as in σ which is possible as we are in the same state and $D_i(\alpha)$ is syntactically taken from α, and was used to produce this trace already during the run that produced σ. Finally, we let the continuous evolution go on until we either hit an evolution domain bound (by β or sampling) or we happen to have evolved for the same amount of time as in σ modulo retiming. This length modulo retiming can be computed as the strategy can already see the choice for s (which is made by Verifier in line 9), the current value of r, and thus knows about the retiming effect. If we are interrupted during a continuous evolution and no discrete action is taken at that time in σ we continue by choosing the "no operation case" in line 5 instead of really performing an action of the program α in line 6, that is

$$?|p_\alpha| = i \wedge \langle t := 0; C_i(\alpha) \wedge H_i(\alpha), \dot{t} = 1\rangle t > 0; p_\alpha := -i$$

which is possible as we know from σ that the loop invariant is not yet violated. We repeat this step multiple times, that is, for each execution of the original loop. Once we reach the end of σ we execute the no-op part of the loop for the remainder of the iterations. This is, the strategy chooses the branch $[?true]$ (line 18).

Using this strategy we get a trace where the sequence of variable valuations of the variables of α is the same as in σ. However, there might be interruptions by β and the retiming control part, and the time axis during continuous evolutions might be stretched or compressed. However, this effect can be reverted using the notion of reverse retiming (Definition 57). Observe that, as the reverse retiming projects the variables of α, in the case where $s = 0$ holds no changes to these variables occur. Thus, the resulting relation only contains a single point up until $r = \varepsilon$ holds. Once, $r = \varepsilon$ is reached the variables evolve with the original speed anyway. The interruptions are not visible under compression either (Definition 54) as no changes to the variables of α occur and the compression function therefore merges these trace elements with their neighbors.

Note that as the strategy v is winning for Verifier the game cannot have ended in a position $(\bot, \varsigma)$. Further, as we have only chosen actions in f that where also taken in σ, we are sure that we also will not end in a position $(\top, \varsigma)$ as $last(\sigma) \neq \Lambda$. Therefore, the play will end in a position

$(\surd, \sigma_p)$. This trace σ_p is by construction a trace equivalent to σ modulo compression and retiming.

Furthermore, the construction of this strategy relied on the auxiliary variables. However, the existence of this strategy does not. Therefore, the results carry over to the original traces and the version of the game that was not annotated. □

Having established a connection between the traces of α and those of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ we now have to create a link to those of β. Therefore, we show that if Verifier follows its winning strategy then whatever strategy Falsifier applies the resulting trace σ_p of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ is equivalent up to retiming to some trace σ_β of β.

Lemma 14. *If there is a winning strategy for Verifier in the game $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$, then there is a Verifier strategy v such that for all Falsifier strategies f holds the trace produced by $\mathfrak{p}_{f,v}(\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta))$ corresponds to a trace of $\tau(\beta)$ with respect to the common variables up to retiming, i.e., for all*

$$(\surd, \sigma) \in \bigcup_{\nu \in Sta(V)} \mathfrak{p}_{f,v}(\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)@\nu)$$

there is $\sigma_\beta \in \tau(\beta)$ such that $ret^{-1}(\sigma, V(\beta), 2-s, r = -\varepsilon \wedge s > 1) \equiv \sigma_\beta$ where s arises in the construction of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ as given in Definition 56.

Proof. We prove the lemma by contradiction. Assume that σ is a trace that was produced by $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ for some strategies f and v, and that no trace $\sigma_\beta \in \tau(\mathfrak{A}(\beta))$ exists that matches this trace. This means that at some point there is a change to the variables of β that is not possible at that point for any trace of β. The trace σ has the form:

$$\begin{aligned}
&\sigma_0, \cdots, \sigma_i, \sigma_{i+1}, \cdots, \sigma_j, \\
&\qquad \sigma_{k,l}, \cdots, \sigma_{m,l}, \sigma_{m+1,l}, \cdots, \sigma_{n,l}, \sigma_{n+1,l}, \sigma_{n+2,l}, \sigma_{n+3,l}, \\
&\sigma_{k,l+1}, \cdots, \sigma_{m,l+1}, \sigma_{m+1,l+1}, \cdots, \sigma_{n,l+1}, \sigma_{n+1,l+1}, \sigma_{n+2,l+1}, \sigma_{n+3,l+1}, \cdots
\end{aligned}$$

Here, indexes 0 to i correspond to the initialization of α, $i+1$ to j to that of β. The indexes (k,l) to (m,l) to the discrete actions of α, the indexes $(m+1,l)$ to (n,l) to the discrete actions of β. However, changes to the variables of β occur using the same syntax as in β itself. Therefore, we can just use the same elements to construct a trace of β that matches σ. During the first execution of the loop we choose the path with matching

values of p_β. When the continuous evolution in the play stops, we analyze the valuations of the variable p_β again. If this variable is negative we continue the current continuous evolution until we match the next end of a continuous evolution in the game. Otherwise, we execute the branch which sets p_β to its current value. This gives us a trace that agrees with σ in the valuations of the variables v_β and thus contradicts our assumptions. □

Using the previous two lemmas we are now able to build a connection between our notion of ε-δ-refinement and our encoding in dDG$\mathcal{L}$. This gives the main theorem for this chapter.

Theorem 9. *If the formula $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)(||v_\alpha - v_\beta|| \leq \delta \wedge |r| \leq \varepsilon)$ is valid for two hybrid programs α and β then α ε-δ-refines β, i.e., $\alpha \wr^{\varepsilon,\delta}\!\rightarrow \beta$.*

Proof. Assume that $\phi \widehat{=} \mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)(||v_\alpha - v_\beta|| \leq \delta \wedge |r| \leq \varepsilon)$ hold and let σ_α be an arbitrary trace of α.

From Lemma 13 we know that this trace can be performed by $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ up to retiming and compression. Furthermore, we know that ϕ holds and, therefore, the distance between the variables of α and those of β never exceeds δ. Further, the temporal deviation is bounded by ε as $|r| \leq \varepsilon$. Now, using Lemma 14 we know that the trace of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ matching σ_α is also matched by a trace of β. As σ_α was arbitrary this holds for all traces of α. Therefore, we can conclude that $\alpha \wr^{\varepsilon,\delta}\!\rightarrow \beta$ holds. □

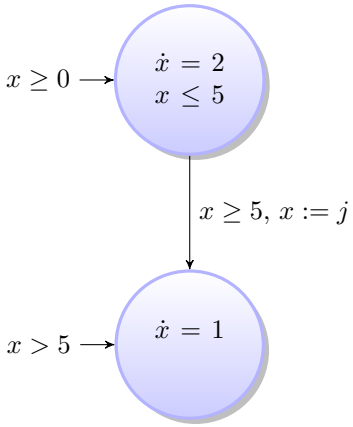
5.2.4 Example

Let j be a constant greater than 5. Consider the following simple example of two hybrid programs α and β also depicted as hybrid automata in Figure 5.8 and Figure 5.10. For system α the value of x is heading towards 5 with a velocity of 2, jumps to j and increases with rate 1 afterwards. The resulting trajectory for initial value $x = 0$ and $j = 8$ is shown in Figure 5.11a. In system β the value of y evolves with speed 1 the whole time and has a single jump from 3 to j. The resulting trajectory for initial value $y = 0$ and $j = 8$ is shown in Figure 5.11b.

We consider the variables x and y to be the same entity and just give them different names for easier reference. Although these two systems differ in their absolute values, when comparing the variables x and y starting in the initial region $x \geq 0$ and $y \geq 0$ they are 0.5-2-similar, i.e., $\alpha \wr^{0.5,2}\!\rightarrow \beta$

$$\alpha \hat{=} \Big(x := *; ?x \geq 0; \Big(\big((?x < 5; v := 2) \cup (?x = 5; x := j; v := 1) \cup (?x > 5; v := 1) \big); x_0 := x; \dot{x} = v \,\&\, (x_0 < 5 \to x \leq 5) \Big)^* \Big)$$

(a) Hybrid program α

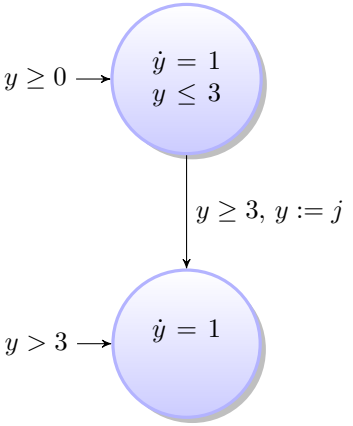


(b) Hybrid automaton α

Figure 5.8: Hybrid system α

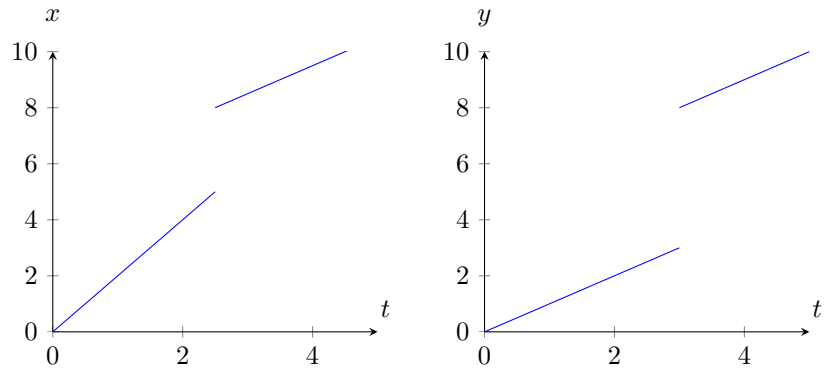
$$\beta \hat{=} \Big(y := *; ?y \geq 0; \Big(\big((?y < 3) \cup (?y = 3; y := j) \cup (?y > 3) \big); y_0 := y; \dot{y} = 1 \,\&\, (y_0 < 3 \to x \leq 3) \Big)^* \Big)$$

(a) Hybrid program β



(b) Hybrid automaton β

Figure 5.10: Hybrid system β



(a) Trajectory of α with $x(0) = 0$ (b) Trajectory of β with $y(0) = 0$

Figure 5.11: Example trajectories with $j = 8$

and $\beta \xleftarrow{0.5,2} \alpha$ both hold. This can be shown by proving that the following invariant holds for the loop occurring in $\mathcal{G}_{0.5,2}(\alpha, \beta)$:

$$x \geq 0 \land y \geq 0 \land \left((x \leq 5 \lor y \leq 3) \rightarrow \left(3x = 5y \land r \leq \frac{1}{6}y\right)\right) \\ \land ((y > 3 \lor x > 5) \rightarrow x = y) \land 0 \leq r \land r \leq 0.5 \quad (5.2)$$

The winning strategy for the Verifier in this game is based on unifying the switching time, i.e., the time where x reaches 5 and y reaches 3. This can be done by using the retiming to ensure that the clock drift ratio of x to y is $\frac{5}{3}$. The retiming that ensures this is constructed by choosing $s = \frac{10}{11}$, which by coordinate transformation gives that $r = \frac{1}{6}y$ before the jump of y. After the jumps, $x = y$ holds and r remains constant and thus in its range of $[0, 0.5]$.

Using the proof rules of Section 4.3 that we implemented in our theorem prover KeYmaera [PQ08a], we can prove that the two systems are similar even without knowing the concrete value of j. Thus, by proving that α and β are similar we did prove this for families of systems depending on the value of j. Our approach is able to deal with multiple parameters and could even yield results for parametric bounds on the temporal and spatial

deviations. Thereby, we demonstrated a huge advantage of using dDG$\mathcal{L}$ over the usual model checking tools as the dDG$\mathcal{L}$-based approach permits parametric reasoning.

5.2.5 Using Existing Properties

Let us take a step back and reconsider the results we just established. Our goal was to transfer properties from a system β to a system α by showing that α robustly refines β. For this, we constructed a game and showed that whenever Verifier has a winning strategy in this game, then α indeed robustly refines β. However, thus far we did not make use of the properties of β that we have established earlier. Observe that, if ϕ is an invariant of β, i.e., $\beta \models \Box_{[0,\infty[}\phi$, then it is an invariant of $\mathcal{G}_{\varepsilon,\delta}(\alpha,\beta)$ as well when interpreted on the variables of β. This means we can add it as an evolution domain constraint, or alternatively as an assertion of the form $[\phi]$ as last statement in the loop.

Of course, also other properties of β can be used as assertions. For those it might be necessary to add additional clocks in order to measure the sets occurring in the formulas as annotations to the until operators.

5.3 Related Work

Even though the games presented in this section are specific to showing ε-δ refinement relations, the idea of using simulation or simulation-like notions in order to show refinement was inspired by Ehrenfreucht-Fraïssé games [Ehr61]. These games are played by two players, usually called Spoiler and Duplicator, on two structures. The goal of Duplicator is to show that these two structures are equivalent while Spoiler tries to disprove this assumption. Examples for these structures are labeled transition systems or first-order structures. On a smaller scale, the goal of Duplicator is to match every action of Spoiler, whereas the goal of Spoiler is to produce a run such that there is an action that cannot be matched by Duplicator. The game is played as follows. In every iteration Spoiler first picks one of the structures and some element in that structure (action for transition systems). Then Duplicator has to match this element (action) on the other structure. If Duplicator cannot match this element then Spoiler wins. Otherwise it is Spoiler's turn again. If Spoiler does not eventually win the game, then we say that Duplicator wins. A strategy for Duplicator

is a list of reactions to every possible move of Spoiler. In case there is a winning strategy for Duplicator then the two structures are equivalent. In the setting of labeled transition system we would have established a bisimulation relation between the two systems. If we fix the structure from which Spoiler chooses his moves it will witness the existence of a simulation instead of bisimulation.

For instance, Tabuada [Tab09] applied this method for hybrid systems. In order to show exact refinement relations and equivalences of systems he defines simulation games. In these games Duplicator controls the implementation and Spoiler controls the specification. This idea strongly relates to the robust refinement games defined in this section. Like in Tabuada's work the games presented in this chapter show stronger properties than what is demanded by refinement. This is necessary as one cannot represent complete hybrid traces in order to pick a matching one from the trace sets of the specification. Instead a matching trace is constructed stepwise. Of course this stepwise construction might miss matching traces as some decisions might be made too early in the construction. Still, when we can show that the stepwise construction succeeds then the two systems are in (robust) refinement relation.

5.4 Conclusion

In this chapter, we have presented two different approaches how to show that a system α is an ε-δ refinement of another system β. The first approach is based on an automaton representation of the systems. It is restricted to the case where the system β is deterministic and we need to explicitly encode the retiming strategy into the automaton. The main advantage of this approach is that once the encoding has been done, the refinement relation can be verified by a model checker automatically.

The second approach is based on our new logic for hybrid games $\mathsf{dDG}\mathcal{L}$. We encode the question whether the systems are in refinement relation in a similar game as we did using automata. In contrast to the automata-based approach, we are not restricted to the class of deterministic systems w.r.t. the choice of system β. In addition, we do not need to explicitly encode the strategy for our retiming. This is an advantage as overall we are only interested in the existence of this strategy in order to be sure that the systems are similar but not in its concrete manifestation.

In addition, we profit from the special design of our logic. That is, we can use the fact that we can encode sequential actions of different players in dDG$\mathcal{L}$ easily and do not have to add artificial committed locations for that. In addition the advance notice semantics of dDG$\mathcal{L}$, i.e., the fact that the number of loop iterations is revealed to the other player in advance, favors this approach. The notion of ε-δ-refinement is defined similar to trace inclusion. Thus, if for each trace σ_α of α the system β can produce a similar one σ_β, then α is considered a robust refinement of β. Here, the choice of the trace σ_β of system β is made based on the whole trace σ_α of system α. However, as already mentioned, we cannot represent the whole trace σ_α. Therefore, the games defined in this chapter are closer to a notion of simulation in the sense that they construct a matching trace of β step by step. This, obviously, is stronger than originally intended. By design, however, dDG$\mathcal{L}$ allows us to reveal additional information about the trace of α. As already mentioned, we use the interleaving semantics on discrete actions to allow choice of discrete transitions of β to be dependent on those just made by α. In addition the advance notice semantics allows us to communicate information about the maximum number of discrete actions early during the game.

Note, that this is one of the main reasons why we decided to use this loop semantics for dDG$\mathcal{L}$ instead of going with a semantics where after each loop iteration the player responsible for controlling the loop may decide to perform the loop body once again.

In addition, dDG$\mathcal{L}$ allows for parametric reasoning. That is, the programs can still contain unspecified variables. Hence the dDG$\mathcal{L}$-based approach allows us to prove robust refinements of classes of systems instead of single systems. Furthermore, we can also use the refinement parameters ε,δ as free variables in our robust refinement game and only specify relation with some system parameters. That way, we could prove refinement relations where the distances could, for instance, depend on the concrete sampling rate of an implementation which occurs freely in its description. Thus, this approach permits to choose the sampling rate later based on which properties the implementation needs to preserve in which way.

CHAPTER SIX

Implementation

Fiction is obliged to stick to possibilities. Truth isn't.
— Mark Twain

Contents

In this chapter, we provide an overview of the implementation of the techniques presented in the previous chapter in our theorem prover KeYmaera. KeYmaera is an interactive theorem prover for differential dynamic logic and its extensions. It is a hybrid theorem prover in a number of aspects.

First of all it is tailored to reason about hybrid systems and hybrid games. Secondly, it uses a combination of powerful proof search strategies with interactive guidance. Thirdly, it combines theorem proving with methods from the domain of computer algebra. Based on the KeY theorem prover [BHS07] for JavaCard DL, a dynamic logic that talks about properties of JavaCard programs, we have implemented KeYmaera by adding support for hybrid programs and interfacing with Mathematica [Wol03] for handling problems of real arithmetic. For this thesis, we added different methods for dealing with real arithmetic and perform a comparison of these in this chapter. These results were partially published in [PQ08a] and [PQR09a].

The proof calculus for dDG$\mathcal{L}$ presented in Chapter 4 has been implemented in KeYmaera and results in proof obligations of the same structure as those coming up in the process of hybrid system verification. Therefore, the study of different methods of dealing with these proof obligations is a worthwhile task.

Contributions. We survey different methods for handling real arithmetic that we implemented in our theorem prover KeYmaera. We present a novel method for dealing with real arithmetic based on semidefinite programming for the Real Nullstellensatz and Gröbner Bases computation that provides checkable certificates. Furthermore, we perform an experimental evaluation of the different methods and implementations.

Structure of the Chapter. In Section 6.1 we sketch the overall approach and the architecture of our theorem prover KeYmaera. Section 6.2 provides a discussion of related approaches. In Section 6.3 we give a brief overview of the techniques implemented in KeYmaera to deal with continuous evolutions. Section 6.4 discusses the topic of handling the resulting first-order formulas over the reals.

6.1 KeYmaera Verification Tool for Hybrid Systems

KeYmaera [PQ08a] is a deductive verification tool for hybrid systems. Originally, we have implemented KeYmaera as a combination of the deductive

theorem prover KeY [BHS07] with the computer algebra system Mathematica [Wol03]. KeY is an interactive theorem prover with a user-friendly graphical interface for proving correctness properties of Java programs. We generalized KeY from discrete systems to hybrid systems by adding support for the differential dynamic logic (d$\mathcal{L}$) [Pla07b, Pla08]. With this, KeYmaera can prove correctness, safety, controllability, reactivity, and liveness properties of hybrid systems. Since then we have severely improved

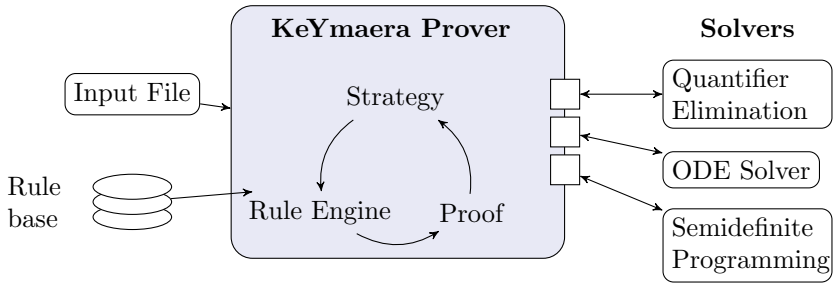


Figure 6.1: Architecture and plug-in structure of the KeYmaera Prover

and extended KeYmaera. We have extended the rule base by implementing rules for handling differential algebraic logic (DAL) [Pla10a, Pla10b], quantified differential dynamic logic (Qd$\mathcal{L}$) [Pla12a], and most importantly for this thesis, differential dynamic game logic (dDG$\mathcal{L}$), our new logic presented in Chapter 4. In addition, we have studied the resulting arithmetic problems and interfaced with a large number of different tools for handling these.

Exploiting the compositional semantics of d$\mathcal{L}$, KeYmaera verifies properties of hybrid programs by proving corresponding properties of their parts in a sequent calculus (cf. Chapter 4 and [Pla07b,Pla08,Pla10b]). KeYmaera uses an automatic strategy for the proof search. In addition interactive guidance by the user is possible and sometimes necessary in order to verify properties of complex systems.

In discrete KeY, rule applications are comparably fast, but in KeYmaera, proof rules that use decision procedures for real arithmetic can require a substantial amount of time to produce a result. To overcome this, we have implemented automatic proof strategies for the hybrid case that navigate among computationally expensive rule applications [Pla10b]. This is, once a sequent only contains first-order formulas there might be several options

on how to proceed with the proof. In theory, the question whether the sequent is valid is decidable by quantifier elimination. Hence a quantifier elimination procedure like it is implemented in Mathematica could be used. However, in practice we noticed that as the computational complexity of this procedure is so high, it is hard to predict whether this will work or not. Alternatively, rules from the propositional sequent calculus could be used to further decompose the verification condition into multiple subtasks. This can lead to tasks that then can be easily handled by quantifier elimination. However, there are problems that split into many thousand equally difficult problems (even equally difficult to the original one). In those cases it is better to apply quantifier elimination early. The iterative background closure procedure suggested in [Pla07a] tries to find the sweetspot of when to apply quantifier elimination.

We have implemented a plug-in architecture for integrating multiple implementations of decision procedures for the different fields of arithmetic handling (cf. Figure 6.1). We integrate arithmetical simplification and real quantifier elimination support by interfacing Mathematica and many other tools. Symbolic solutions of differential equations, which can be used for handling continuous dynamics, are obtained either from Mathematica or Orbital, a math library for Java developed by André Platzer. Note that, the architecture of KeYmaera is designed in such a way that other tools for this task can be added conveniently.

6.2 Alternative Approaches

Theorem proving based approaches have been used for verifying hybrid systems in STeP [MS98] or PVS [ÁMSH01]. However, they do not use a genuine logic for hybrid systems but compile prespecified invariants of hybrid automata into an overall verification condition. Furthermore, by using background solvers and iterative background closure strategies, we obtain a larger degree of automation than interactive proving in STeP [MS98] or higher-order logic [ÁMSH01]. VSE-II [NRS01] uses discrete approximations of hybrid automata for verification. In contrast, KeYmaera respects the full continuous-time semantics of hybrid systems. HyTech [HHWT97], PHAVer [Fre05], SpaceEx [FGD$^+$11], and FOMC [DDD$^+$12] are model checkers primarily for linear hybrid automata. SpaceEx and FOMC further feature methods to deal with linear differential equations. CheckMate [SRKC00] supports more complex continuous dynamics, but still re-

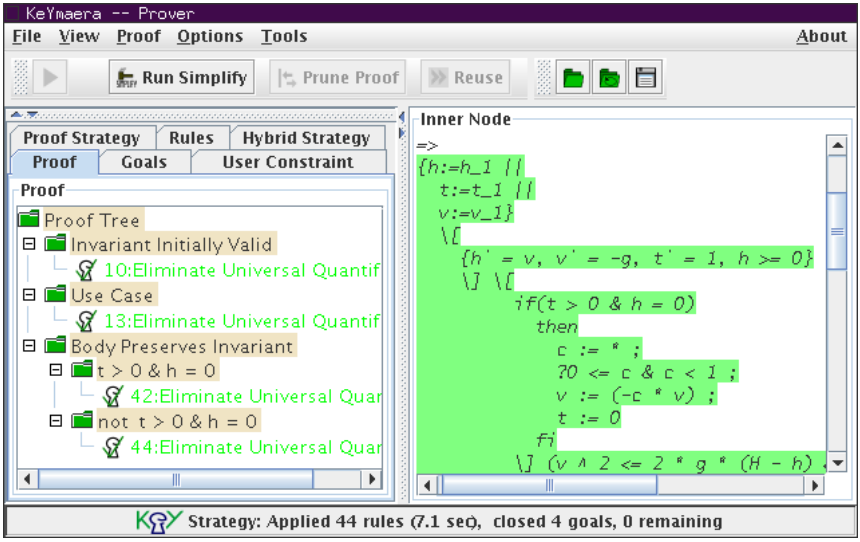


Figure 6.2: Screenshot of the KeYmaera user interface

quires initial states and switching surfaces to be linear. KeYmaera supports nonlinear constraints on the system parameters as required for train, car, or airplane applications or even simple examples like the parametric bouncing ball.

Recently, Renshaw did a prototypical partial reimplementation of our theorem prover KeYmaera, called KeYmaeraD [RLP11]. It features interesting ideas w.r.t. the representation of proof search strategies and parallel proof search. However, it does only cover the universal fragment of differential dynamic logic (d$\mathcal{L}$), differential algebraic dynamic logic (DAL), and quantified differential dynamic logic (Qd$\mathcal{L}$), i.e., it does not allow for any diamond modalities. In the settings of dDG$\mathcal{L}$ diamond modalities are crucial as only these allow for specifying actions under the control of Verifier. In addition, KeYmaeraD only interfaces with Mathematica and Renshaw's own implementation of Cohen-Hörmander quantifier elimination for dealing with real arithmetic whereas KeYmaera use a variety of different approaches (cf. Section 6.4).

6.3 Handling of Differential Equation Systems

For handling continuous evolutions we implement two different approaches. The first approach is based on solutions of differential equation systems implementing the rules presented in Chapter 4. That is, we implement the rules D11 and D12 (see page 102). In order to generate the required solutions to the differential equation systems, we use Mathematica and the Orbital library.

As an alternative approach we have implemented differential induction based on the work of Platzer on differential algebraic logic (DAL) [Pla10a, Pla10b]. The main idea is to strengthen evolution domain constraints. For that, we prove that in the region described by the current evolution domain constraint the gradient of the system points inwards w.r.t. the region described by the differential invariant candidate. This is much like classical induction used to prove loop invariants. However, in this case the effect of the program is not described in terms of discrete assignments, tests, and choices but rather by derivatives of system variables. When proving loop invariants, we show that after one execution of the loop body the invariant is preserved. For continuous evolution there is no discrete notion of progress. Thus, we have to show that at every point within the evolution domain the trajectories point inwards w.r.t. to our differential invariant candidate. We refer the reader to [Pla12c,Pla10b] for more details on this issue.

6.4 Dealing with Arithmetic

We have implemented various ways of dealing with the arithmetic formulas resulting from the decomposition of hybrid games and programs. The calculus presented in Chapter 4 decomposes a given formula into a set of verification conditions formulated in first-order logic over the reals. From the soundness of the calculus it follows that if all these are valid, then the original formula is valid.

The arithmetic formulas are formulas in first-order logic over the reals. In the early 20th century, Tarski [Tar51] proved that this logic is decidable in the sense that it is equivalent to the first-order theory of real-closed fields, which is decidable by quantifier elimination. This is, one can replace quantified formulas equivalently by quantifier-free formulas. Hence we ultimately end up with a propositional variable-free formula containing terms

with only rational constants and $+, -, \cdot$ as function symbols connected by the usual predicates for real arithmetic.

For quantifier elimination [CH91, Wei97] various implementations exists. We have implemented generic interfaces within KeYmaera and refined these tools.

In addition, we studied approaches based on Gröbner Bases [Buc65], and semidefinite programming [Par03] for the Positivstellensatz [Ste73] for the universal fragment of first-order logic over the reals. We interface with modern satisfiability modulo theory (SMT) solvers, as these also implement techniques to handle real arithmetic.

In [PQR09a], we presented a new decision procedure for the universal fragment of real-closed fields that combines Gröbner Basis computations with semidefinite programming for the real Nullstellensatz [Ste73] to avoid the scalability issues with semidefinite programming for the Positivstellensatz.

Overall, we compare the following approaches:

1. Quantifier elimination for real-closed fields in Mathematica [Wol03], QEPCAD B [Bro03], Redlog [DS97], HOL Light [MH05], and Renshaw's implementation of the Cohen-Hörmander procedure directly in KeYmaera;

2. Real arithmetic handling with Gröbner Bases using external procedures in Mathematica, the Orbital library, and internally with KeYmaera proof rules;

3. Semidefinite programming relaxations [Par03] for the Positivstellensatz [Ste73] using the CSDP solver [Bor99] in our own implementation and in HOL Light [Har07];

4. Our new algorithm combining Gröbner Bases and semidefinite programming for the real Nulstellensatz [Ste73] using CSDP [Bor99] and the Orbital library.

5. The SMT solver Z3 [dMB08], implementing an approach based on Boolean satisfiability (SAT) modulo real-arithmetic [JdM12] where the search is guided by the projection operators originally defined for cylindrical algebraic decomposition [Col75], a method for real quantifier elimination.

Note that the following sections are taken from joint work with André Platzer and Philipp Rümmer with only minor revisions compared to the work previously published in [PQR09a].

6.4.1 Methods for Handling Real Arithmetic

We survey different approaches for handling real arithmetic in background provers for verification following [PQR09a]. We phrase these approaches in terms of reals for simplicity. Yet, all subsequent theory in Sections 6.4.1–6.4.2 generalizes from $\mathbb{R}$ to real-closed fields.

To simplify the presentation, we assume simple rules to normalize sequents that translate, e.g., $g \leq f$ to $f \geq g$, $f \neq g$ to $\neg(f = g)$ and $\vdash f > g$ to $f \leq g \vdash$ respectively. These rules are shown in Figure 6.3. We assume

$$\text{(N1)}\ \frac{\Gamma, f \geq g \vdash f = g, \Delta}{\Gamma, g < f \vdash \Delta} \qquad \text{(N5)}\ \frac{\Gamma, f \geq g \vdash \Delta}{\Gamma, g \leq f \vdash \Delta}$$

$$\text{(N2)}\ \frac{\Gamma, f \geq g \vdash \Delta}{\Gamma \vdash f < g, \Delta} \qquad \text{(N6)}\ \frac{\Gamma, f \geq g \vdash f = g, \Delta}{\Gamma \vdash f \leq g, \Delta}$$

$$\text{(N3)}\ \frac{\Gamma, f \geq g \vdash f = g, \Delta}{\Gamma, f > g \vdash \Delta} \qquad \text{(N7)}\ \frac{\Gamma, f \geq g \vdash f = g, \Delta}{\Gamma \vdash g \geq f, \Delta}$$

$$\text{(N4)}\ \frac{\Gamma, f \geq g \vdash \Delta}{\Gamma \vdash g > f, \Delta}$$

$$\text{(N8)}\ \frac{\Gamma, f = g \vdash \Delta}{\Gamma \vdash f \neq g, \Delta}$$

$$\text{(N9)}\ \frac{\Gamma \vdash f = g, \Delta}{\Gamma, f \neq g \vdash \Delta}$$

Figure 6.3: Rules for normalizing equalities and inequalities

all inequalities to be moved to the antecedent in this way. We get 4 rules for handling "greater than" ($>$) and "less than" ($<$) as we want to eliminate these operators. For "greater or equals" ($\geq$) we only have to consider the

case where it occurs on the right side of the sequent. Whereas, for "less or equals" ($\leq$) we get two cases. In addition we have two cases for "not equals" ($\neq$).

6.4.1.1 Gröbner Bases for Real Arithmetic

Gröbner bases [Buc65] provide a sound but incomplete procedure for proving validity of formulas in the universal fragment of equational first-order real arithmetic.

Preliminaries. Let $\mathbb{Q}[X_1, \ldots, X_n]$ be the set of *multivariate polynomials* over the indeterminates $X_1, \ldots, X_n$ with coefficients in $\mathbb{Q}$. Each of these polynomials p can be written as

$$p = \sum_i c_i X_1^{d_1(i)} \cdots X_n^{d_n(i)} \ ,$$

where $c_i \in \mathbb{Q}$ are coefficients and $d_j(i) \in \mathbb{N}$ for $j \in \{1, \ldots, n\}$. Terms of the form $c_i X_1^{d_1(i)} \cdots X_n^{d_n(i)}$ are called *monomials*. A subset $I \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ is an *ideal*, iff I is a subgroup with respect to addition and

$$rx \in I, \text{ for all } x \in I, r \in \mathbb{Q}[X_1, \ldots, X_n] \ .$$

The ideal *generated* by a set $G \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ is the smallest ideal I containing G, and is denoted by (G).

The notions of Gröbner bases and polynomial reductions are relative to an admissible monomial order $\prec$, which is a strict well-order on monomials such that $uw \prec vw$ whenever $u \prec v$ for arbitrary monomials u, v, w. Admissible orders extend canonically to $\mathbb{Q}[X_1, \ldots, X_n]$ as a multiset order; see [Buc65] for details. The monomial order determines the *leading term* in multivariate polynomials, i.e., the maximal monomial with respect to $\prec$.

Definition 59 (Reduction)**.** *Let $f, g \in \mathbb{Q}[X_1, \ldots, X_n]$. We say that f* reduces to *g with respect to $G \subset \mathbb{Q}[X_1, \ldots, X_n]$ iff for some $m \in \mathbb{N}$ there are $f_0, f_1, \ldots, f_m$ in $\mathbb{Q}[X_1, \ldots, X_n]$ with $f_0 = f, f_m = g$ such that, for all i, $f_{i+1} = f_i - h_i g_i$ for some $h_i \in \mathbb{Q}[X_1, \ldots, X_n]$, $g_i \in G$, and $f_{i+1} \prec f_i$. We write $g = \mathrm{red}_G f$ if, in addition, g cannot be reduced further, i.e., there is no $h_{m+1} \in \mathbb{Q}[X_1, \ldots, X_n]$ and $g_{m+1} \in G$ with $g - h_{m+1}\, g_{m+1} \prec g$.*

Definition 60 (Gröbner basis)**.** *A finite subset G of an ideal I of the polynomial ring $\mathbb{Q}[X_1, \ldots, X_n]$, is called* Gröbner basis *iff $I = (G)$ and* $\mathrm{red}_G f$ *is unique for all polynomials f. Further, G is* reduced *if no $g \in G$ can be reduced further with respect to $G \setminus \{g\}$.*

There are several equivalent alternative formulations of this definition, for instance that $\mathrm{red}_G f = 0$ iff $f \in I$. This means that Gröbner bases solve the *ideal membership problem* and can, thus, directly be used as an (incomplete) proof rule for equational arithmetic.

Gröbner Basis Eliminations. The most naive use of Gröbner bases for real arithmetic is described by the rules A1, A2 in Figure 6.4. The rule A1 closes a goal if the ideal G generated by equations in the antecedent contains 1. The soundness of this rule relies on Hilbert's Nullstellensatz.

Theorem 10 (Hilbert's Nullstellensatz [Hil93])**.** *For an algebraically closed field K and some ideal $I \subsetneq K[X_1, \ldots, X_n]$ there exists some vector x with $x = (x_1, \ldots, x_n) \in K^n$ such that $p(x_1, \ldots, x_n) = 0$ for all $p \in I$.*

This means if the constant function 1 is in the ideal (G), i.e., $1 \in (G)$, then the equations do not have common solutions (i.e., are contradictory). Note that this fact is crucial for the understanding of this chapter as the many of the upcoming methods rely on detecting a contradiction in the antecedent in this way. Similarly, A2 can be applied if the sides f, h of an equation in the succedent have the same remainder modulo G, which means $f - h \in (G)$.

The scope of the rules can be extended by testing for *radical membership* instead of ideal membership, which can prove problems like $x^2 = 0 \vdash x = 0$ that A2 cannot prove. The *radical* of an ideal I is the set

$$\sqrt{I} = \bigcup_{i=1}^{\infty} \{g \in \mathbb{Q}[X_1, \ldots, X_n] \mid g^i \in I\} \supseteq I$$

Because the inclusion $I \subseteq \sqrt{I}$ can be strict, testing for radical membership is more liberal than ideal membership, while still being sound.

Example 22. *Consider the ideal (x^2) generated by the polynomial x^2. This contains all squares and sums of squares. Observe that, it does not contain any polynomials with odd degrees like x, x^3, x^5. The radical $\sqrt{(x^2)}$, however, contains all polynomials from (x), i.e., $\sqrt{(x^2)} = (x)$.*

In practice, the rule A3, which is known as *Rabinowitch's trick*, represents a simple way of testing for radical membership. It is based on the observation that $g \in \sqrt{I}$ if and only if $1 \in (I \cup \{gz - 1\})$ (where z is a fresh indeterminate). The latter property can be tested by first applying A3 and then A1.

Finally, inequalities can be translated to equations using A4, A5, which exploit the fact that a real number is positive iff it is a square (A5 is an optimized version including Rabinowitch's trick). Combined with the rules A1, A2, this encoding of inequalities is rather weak, and not able to derive simple facts like $(a \leq b \wedge b \leq c) \rightarrow a \leq c$. It is, however, an important preprocessing step for the complete procedure described in the next subsection (where we explain rule A6).

Proposition 8 (Soundness [PQR09b])**.** *The Gröbner basis rules in Figure 6.4 are sound. Rules A3, A4, A5 are even satisfiability-equivalent transformations, i.e., their respective premisses and conclusions are satisfiability-equivalent.*

Proof. The rules in Figure 6.4 are sound. As usual for the soundness proofs we assume Γ to be true in a state ν and Δ to be false as there is nothing to show otherwise. For A3, A4, A5, we show equivalence of premiss and conclusion, which implies soundness.

A2 Suppose the conclusion was false in ν, i.e.,

$$\nu \models g_1 = \tilde{g}_1 \wedge \cdots \wedge g_n = \tilde{g}_n \wedge f \neq h \ .$$

Thus, $\nu \models g = 0$ for all $g \in G$. Consequently, $\nu \models g = 0$ for all polynomials g in the ideal (G) of G. As a consequence of the applicability condition, we have $\mathrm{red}_G(f - h) = 0$, which, by Definition 60, implies that $f - h$ is in the ideal of G. In combination, we have $\nu \models f - h = 0$, hence $\nu \models f = h$, which is a contradiction.

A1 The soundness of A1 is a special case of the soundness of A2 when assuming the false formula $1 = 0$ for the succedent $f = h$.

A3 Satisfiability-equivalence of A3 is a consequence of the Rabinowitch trick equivalence

$$x \neq 0 \leftrightarrow \exists z \, (xz = 1) \ ,$$

using $f - g$ for x. More generally, this holds in fields where nonzero elements are exactly the elements that have some inverse z.

$$\text{(A1)}\quad \frac{*}{\Gamma, g_1 = \tilde{g}_1, \ldots, g_n = \tilde{g}_n \vdash \Delta}$$

$$\text{(A2)}\quad \frac{*}{\Gamma, g_1 = \tilde{g}_1, \ldots, g_n = \tilde{g}_n \vdash f = h, \Delta}$$

$$\text{(A3)}\quad \frac{\Gamma, (f-g)z = 1 \vdash \Delta}{\Gamma \vdash f = g, \Delta}$$

$$\text{(A4)}\quad \frac{\Gamma, f - g = z^2 \vdash \Delta}{\Gamma, f \geq g \vdash \Delta}$$

$$\text{(A5)}\quad \frac{\Gamma, (f-g)z^2 = 1 \vdash \Delta}{\Gamma, f > g \vdash \Delta}$$

$$\text{(A6)}\quad \frac{\Gamma \vdash 1 + s_1^2 + \cdots + s_n^2 = 0, \Delta}{\Gamma \vdash \Delta}$$

In all rules, z is a fresh variable. With the Gröbner basis G of the ideal $(g_1 - \tilde{g}_1, \ldots, g_n - \tilde{g}_n)$, rule A1 is applicable if $\mathrm{red}_G\, 1 = 0$, and A2 if $\mathrm{red}_G\, f = \mathrm{red}_G\, h$. Rules similar to A2, A4 and A5 can be defined for inequalities in the succedent. In A6, the polynomials $s_1, \ldots, s_n$ can be chosen arbitrarily.

Figure 6.4: Rule schemata of Gröbner calculus rules

By introducing a free new variable z, we obtain that the premiss is satisfiable if and only if the conclusion is satisfiable.

A4 Satisfiability-equivalence follows from the equivalence

$$f \geq g \leftrightarrow \exists z\, (f - g = z^2)$$

in the domain of reals. More generally, this holds in real-closed fields where squares are exactly the positive numbers. By introducing a free new variable z in the rule, we obtain that the premiss and conclusion are satisfiability-equivalent, i.e., the premiss is satisfiable if and only if the conclusion is satisfiable.

A5 Satisfiability-equivalence follows from the equivalence

$$x > 0 \leftrightarrow \exists z\,(xz^2 = 1)$$

in the reals, using $f - g$ for x.

A6 Since a sum of squares is nonnegative over $\mathbb{R}$, the value of the polynomial $1 + s_1^2 + \cdots + s_n^2$ is strictly positive and $1 + s_1^2 + \cdots + s_n^2 = 0$ is a contradiction over the reals. Consequently, if the premiss is valid, then so is the conclusion. □

The Gröbner basis approach gives a sound but incomplete overapproximation. To see why Gröbner bases are incomplete for real arithmetic, consider the following. Gröbner bases are a general approach and do not take into account the special properties of the reals. For instance, the sequent $x^2 = -1 \vdash$ is valid, i.e., the formula $x^2 = -1$ is unsatisfiable over $\mathbb{R}$, but the Gröbner basis of $x^2 + 1$ is $\{x^2 + 1\}$ and, in fact, $x^2 = -1$ is satisfiable over the field of complex numbers $\mathbb{C}$ but not over the field of real numbers $\mathbb{R}$. Therefore, we can not detect the contradiction in the assumption that $x^2 = -1$ using Gröbner bases.

Implementations. We compare three implementations of the Gröbner basis rules:

GM The implementation provided by the Mathematica 9.0 computer algebra system [Wol03], which can be used as a reasoning back-end by KeYmaera.

GO The implementation of Buchberger's algorithm [Buc65] in the open-source Java-library Orbital (written by André Platzer).

GK An implementation of Buchberger's algorithm with (verified) proof rules that are directly defined within KeYmaera. This procedure generalizes a calculus for integer arithmetic [Rüm07] to the reals, and differs from GM and GO in that it does not use the rules A3, A4, A5, but instead integrates the Fourier-Motzkin variable elimination rule [Sch86] to handle inequalities (which is complete for linear arithmetic). This tight integration of the two procedures can simplify terms in inequalities using Gröbner bases, and can feed equations derived by the Fourier-Motzkin procedure back to Buchberger's algorithm. We evaluate the benefits of this cooperation in Section 6.4.3. Since our domain are the reals, we do

not use the heuristic approach tailored to nonlinear integer inequalities from [Rüm07].

6.4.1.2 A Complete Rule using the Real Nullstellensatz

While the rules A1, A2, A3, A4, A5 only form an incomplete calculus for problems in real arithmetic, the situation is different over the complex numbers: Hilbert's Nullstellensatz tells that A1, A3 together yield a decision procedure for universal equational problems in algebraically closed fields like $\mathbb{C}$. A corresponding result for real-closed fields is Stengle's real Nullstellensatz [Ste73]; also see [Har07]:

Theorem 11 (Nullstellensatz [Ste73] for real-closed fields)**.** *Let R be a real-closed field (for instance, $R = \mathbb{R}$) and G be a finite subset of $R[X_1, \ldots, X_n]$. Then the set $\{x \in R^n \mid g(x) = 0 \text{ for all } g \in G\}$ is empty if and only if there are $s_1, \ldots, s_m \in R[X_1, \ldots, X_n]$ such that $1 + s_1^2 + \cdots + s_m^2 \in (G)$. If, moreover, $G \subseteq \mathbb{Q}[X_1, \ldots, X_n]$, then also the polynomials $s_1, \ldots, s_m$ can be chosen among the elements of $\mathbb{Q}[X_1, \ldots, X_n]$.*

This theorem leads to an extremely simple, yet complete, proof method for the universal fragment of real arithmetic: in addition to the rules that we have already discussed, we add a proof rule A6 in Figure 6.4 for injecting the equation $1 + s_1^2 + \cdots + s_m^2 = 0$ into the succedent a proof goal. Any valid proof goal can then be closed in the following way: (*i*) inequalities and equations in the succedent are turned into equations in the antecedent with the help of the rules A3, A4, and A5, (*ii*) the *witness* $1 + s_1^2 + \cdots + s_m^2$ due to the real Nullstellensatz is generated using A6, and (*iii*) the goal is closed by the Gröbner Basis computations with A2.

Corollary 8 (Completeness)**.** *Along with the propositional rules in Figure 4.6 and normalization rules in Figure 6.3, the rules in Figure 6.4 are complete for the universal fragment of real arithmetic.*

Proof. Completeness follows from Theorem 11 using the satisfiability-equivalence properties for the transformation by A3, A4, A5 according to Proposition 8. □

The main difficulty with this calculus is obvious: it does not provide any guidance for choosing the witness $1 + s_1^2 + \cdots + s_m^2 = 0$. One technique to tackle the required search is semidefinite programming, following the work based on Stengle's Positivstellensatz (Section 6.4.1.5) in [Par03, Har07].

$$\dfrac{\dfrac{\dfrac{}{x-y=a^2, z=b^2, (yz-xz)c^2=1 \vdash 1+(abc)^2=0}\,*}{x-y=a^2, z=b^2, (yz-xz)c^2=1 \vdash}\,\text{A6}}{x \geq y, z \geq 0, yz > xz \vdash}\,\text{A2}\ \text{A4,A5}$$

Figure 6.5: Example proof using the real Nullstellensatz

Further, we describe an approach that combines semidefinite programming with Gröbner bases in Section 6.4.2 that was first presented in [PQR09a].

Example 23. *In Figure 6.5, we show a proof for the following implication (leaving out propositional reasoning):*

$$(x \geq y \wedge z \geq 0) \rightarrow xz \geq yz. \tag{6.1}$$

The inequalities $x \geq y$ and $z \geq 0$ are turned into equations using A4. Proving by contradiction (or using propositional rules), the conclusion $xz \geq yz$ is considered as an assumption $yz > xz$ and subsequently eliminated with the help of A5. Once this is done, we rely on an oracle to tell us the witness $1+(abc)^2$, which is introduced using A6. Finally, the proof can be closed by A2: the set $\{a^2 - x + y,\, b^2 - z,\, xzc^2 - yzc^2 + 1\}$ is a Gröbner basis representing the equations in the antecedent. The basis reduces the term $1+(abc)^2$ to 0 as follows:

$$1 + a^2b^2c^2 \overset{b^2-z}{\rightsquigarrow} 1 + a^2zc^2 \overset{a^2-x+y}{\rightsquigarrow} 1 + xzc^2 - yzc^2 \rightsquigarrow 0$$

6.4.1.3 Quantifier Elimination in Real-Closed Fields

A general method for handling quantified real arithmetic is based on the seminal work by Tarski [Tar51]. He showed that there is an algorithm computing a quantifier-free formula that is equivalent to a given formula in (first-order) real arithmetic.

Theorem 12 (Quantifier elimination [Tar51])**.** *The first-order theory of reals (or of real-closed fields) admits* quantifier elimination*, i.e., to each first-order formula ϕ, a quantifier-free formula* $\mathrm{QE}(\phi)$ *can be associated effectively that is equivalent and has no additional free variables. Thus* QE *yields a decision procedure for closed formulas when evaluating the remaining quantifier-free formulas.*

Unlike the other approaches outlined in this chapter, QE directly applies to full nonlinear (polynomial) real arithmetic and not just to the universal fragment. QE is also independent of propositional rules, except that computational efficiency considerations advise to combine both [Pla07a].

Example 24. *For instance,* QE *yields the following equivalence:*

$$\exists x\,(ax^2 + bx + c = 0) \;\equiv\; a \neq 0 \wedge b^2 - 4ac \geq 0 \vee a = 0 \wedge (b = 0 \rightarrow c = 0)$$

Tarski's approach is of non-elementary complexity. However, it has been extended to practical algorithms [Col75, CH91, Wei97], which are quite sophisticated.

Collin's [Col75] introduced cylindrical algebraic decomposition. Let

$$\phi \mathrel{\hat{=}} Q_1 x_1 \;\cdots\; Q_n x_n \;\psi$$

for $Q_i \in \{\forall, \exists\}$ be a first-order formula over the reals in prenex form, i.e., ψ is quantifier free. Thus ϕ has n variables. Collin's idea is to decompose the n-dimensional real-space into a finite number of disjoint connected sets (called cells). In each such cell all the polynomials are sign-invariant, i.e., the sign of each polynomial in ψ is either non-negative or non-positive at each point in the set. One can now determine in which of these cells the quantifier-free formula ψ is true as the decomposition ensures that it is sufficient to check a single point for each cell. This allows to regard universal and existential quantifiers to be seen as conjunctions and disjunctions as they only bind finitely many distinguishable elements w.r.t. the equivalence relation induced by the decomposition into cells. Collin's method constructs the cells in a way such that they can then be described in terms of polynomials in $n-1$ variables and, thus, the quantifiers can be eliminated one by one. Overall, this method is doubly exponential in the number of variables [Col75].

Several optimizations where proposed for this method [CH91, Wei97]. Unfortunately, the complexity of QE is doubly exponential in the number of quantifier alternations [DH88].

Implementations. We compare seven implementations of QE in experiments:

QQ Partial cylindrical algebraic decomposition (PCAD) [CH91] in QEPCAD B [Bro03];

QQ$_r$ Redlog [DS97] using QQ after simplifications;

QM QE based on partial CAD [CH91] and validated numerics [Str06] in Mathematica [Wol03];

QR Virtual substitution [Wei97] in Redlog [DS97], falling back to an internal partial CAD implementation [CH91];

QR$_q$ Virtual substitution [Wei97] in Redlog [DS97], falling back to QQ;

QC Harrison's implementation of Cohen-Hörmander quantifier elimination;

QK David Renshaw's implementation of Cohen-Hörmander quantifier elimination directly in KeYmaera;

QH Proof-producing quantifier elimination [MH05] in HOL Light.

6.4.1.4 Boolean Satisfiability Modulo the Theory of Real-closed Fields

Jovanović and de Moura [JdM12] recently presented an approach based on Boolean satisfiability modulo theory (SMT) that in contrast to the other approaches tries to construct a model instead of showing its absence. Thus, from our point of view we start with the negation $\neg F$ of the formula F which we want to prove valid. That way, if a model can be constructed for $\neg F$ we know that our assumption that F would be valid was false. Otherwise, if their approach can come up with a proof why no model for $\neg F$ can be found, we are certain that the original formula F is indeed a valid.

Jovanović and de Moura's method performs a backtracking search for a model. When a conflict in the sense of Conflict-Driven-Clause Learning (CDCL) [SS99] is discovered they perform a projection into lower-dimensional spaces similar to CAD on the set of the conflicting formulas.

These ideas have been implemented in the SMT solver **Z3** [dMB08] which we have connected as a back-end procedure to KeYmaera.

6.4.1.5 Semidefinite Programming for the Positivstellensatz

The Positivstellensatz for real-closed fields [Ste73] is a generalisation of the real Nullstellensatz. It gives rise to a sound and complete proof method for the universal fragment of first-order real arithmetic that does not require the reductions A3, A4, A5. The Positivstellensatz has recently been exploited in combination with relaxations from semidefinite programming by Parrilo [Par03] and Harrison [Har07].

For this, we need in addition to ideals the following algebraic structures: monoids and cones.

Definition 61 (Multiplicative Monoid)**.** *For a set $H \subseteq R[X_1, \ldots, X_n]$ of polynomials, the* multiplicative monoid $\mathrm{mon}(H)$ *is the smallest set such that:*

1. $1 \in \mathrm{mon}(H)$
2. *If* $p \in H$ *then* $p \in \mathrm{mon}(H)$.
3. *If* $p, q \in \mathrm{mon}(H)$ *then* $p \cdot q \in \mathrm{mon}(H)$.

Definition 62 (Cone)**.** *For a set $F \subseteq R[X_1, \ldots, X_n]$ of polynomials, the* cone $\mathrm{con}(F)$ *is the smallest set such that:*

1. *If* $p \in F$ *then* $p \in \mathrm{con}(F)$.
2. *If* $p \in R[X_1, \ldots, X_n]$ *then* $p^2 \in \mathrm{con}(F)$.
3. *If* $p, q \in \mathrm{con}(F)$ *then* $p + q \in \mathrm{con}(F)$ *and* $p \cdot q \in \mathrm{con}(F)$.

Observe that the cone is not only generated from the polynomials in F but contains all squares of polynomials of the original polynomial ring. For more computational representations of cones and ideals, we refer to [Par03, BCR98].

Theorem 13 (Positivstellensatz [Ste73] for real-closed fields)**.** *Let R be a real-closed field (for instance, $R = \mathbb{R}$) and F, G, H finite subsets of the polynomial ring $R[X_1, \ldots, X_n]$. Then*

$$\{x \in R^n \mid f(x) \geq 0 \text{ f.a. } f \in F,\ g(x) = 0 \text{ f.a. } g \in G,\ h(x) \neq 0 \text{ f.a. } h \in H\}$$

is empty iff

$$\text{there are } s \in \mathrm{con}(F), g \in (G), m \in \mathrm{mon}(H) \text{ such that } s + g + m^2 = 0\ .$$

$$\text{(A7)}\quad \frac{*}{f_1 \geq \tilde{f}_1, \ldots, f_m \geq \tilde{f}_m, g_1 = \tilde{g}_1, \ldots, g_n = \tilde{g}_n \vdash h_1 = \tilde{h}_1, \ldots, h_l = \tilde{h}_l}$$

A7 is applicable iff $s + g + m^2 = 0$ for some $s \in \mathrm{con}(\{f_1 - \tilde{f}_1, \ldots, f_m - \tilde{f}_m\})$, some $g \in (g_1 - \tilde{g}_1, \ldots, g_n - \tilde{g}_n)$, and some $m \in \mathrm{mon}(\{h_1 - \tilde{h}_1, \ldots, h_l - \tilde{h}_l\})$.

Figure 6.6: Rule schemata of Positivstellensatz calculus rules

If, moreover, $F, G, H \subseteq \mathbb{Q}[X_1, \ldots, X_n]$, then also the polynomials s, g, m can be chosen among the elements of $\mathbb{Q}[X_1, \ldots, X_n]$.

The polynomials s, g, m are polynomial infeasibility witnesses. Provided a bound on the degree, witnesses s, g, m can be searched for using numerical semidefinite programming [Par03] by parameterizing the resulting polynomials. As (theoretical) degree bounds exist for the certificate polynomials s, g, m, the Positivstellensatz yields a decision procedure. These bounds are, however, at least triply exponential [Par03]. Thus, the approach advocated by Parrilo [Par03] is to increase the bound successively and solve the existence of bounded degree witnesses due to the Positivstellensatz by semidefinite programming [BV04].

As a simple corollary to Theorem 13 we have that A7 is a sound proof rule.

Corollary 9 (Soundness)**.** *The rule in Figure 6.6 is sound.*

In contrast to the rules in Figure 6.4 the only additional transformation necessary for rule A7 is a reduction from $>$ to $\geq$ via $f > g \leftrightarrow f \geq g \wedge f \neq g$. All other transformations follow from the propositional sequent calculus rules (see Figure 4.6 on page 101) and the rewriting rules described in the beginning of Section 6.4.1 (see Figure 6.3). Therefore, this approach does not introduce new variables, as it does not need the rules A3 – A5. Alternatively, A5 can be used in place of the $f > g$ axiomatization as we show in the sequel.

Example 25. *A proof for the implication* (6.1) *that uses the Positivstellensatz is depicted in Figure 6.7. In contrast to the proof in Figure 6.5, it is now unnecessary to eliminate the inequalities $x \geq y$ and $z \geq 0$, while*

the rule A5 has to be used for $xz \geq yz$ (corresponding to $yz > xz$ in the antecedent). A witness for the problem is:

$$\underbrace{c^2 \cdot (x-y) \cdot z}_{s} + \underbrace{(yz-xz)c^2 - 1}_{g} + \underbrace{1}_{m^2} = 0$$

The terms $x - y$ and z in s stem from the inequalities in the sequent, while the term g is derived from the equation.

Implementations. We compare two implementations using the semidefinite programming optimization tool CSDP [Bor99] to find witnesses for the Positivstellensatz:

PH John Harrison's implementation [Har07] in HOL Light.

PK Our implementation within KeYmaera directly follows the approach presented by Parrilo [Par03] and Harrison [Har07]. We follow Parrilo's enumeration of polynomials without further optimization.

6.4.2 Gröbner Bases for the Real Nullstellensatz (GRN)

We present an approach to turn the complete calculus based on the real Nullstellensatz (NSS, Theorem 11) into an effective proof procedure. While this method is strongly inspired by, and in parts based on, semidefinite programming for the Positivstellensatz (PSS, Theorem 13) [Par03, Har07], there are two main motivations to deviate from this approach: (*i*) the application of the PSS requires reasoning about ideal membership (the set (G) in Theorem 13) and, thus, to solve systems of polynomial equations. This is an incentive to integrate Gröbner bases as a computational, efficient, and well-studied method to this end; (*ii*) the PSS requires constructing three witnesses s, g, m simultaneously, which makes it intricate to balance

$$\dfrac{\dfrac{}{x \geq y, z \geq 0, (yz - xz)c^2 = 1 \vdash}\,{}^{*}\,\text{A7}}{x \geq y, z \geq 0, yz > xz \vdash}\,\text{A5}$$

Figure 6.7: Example proof using the Positivstellensatz

degree bounds and the number of parameters to be determined by semidefinite programming. Using a combination of Gröbner basis computations and the single witnesses of the real NSS, these issues are avoided.

In order to prove by NSS that a set G of polynomials does not have common zeroes, we need to find polynomials $s_1, \ldots, s_m$ such that the polynomial $1 + s_1^2 + \cdots + s_m^2 \in (G)$. We reduce this problem to a search for positive semidefinite matrices with the help of the following lemma. A matrix $X \in \mathbb{R}^{k \times k}$ is called *positive semidefinite* (PSD) if it is symmetric, and if $x^t X x \geq 0$ for each vector $x \in \mathbb{R}^k$. There is a simple correspondence between PSD matrices and sums of squares:

Lemma 15. *Suppose $p \in \mathbb{Q}[X_1, \ldots, X_n]^k$ is a vector of rational polynomials. The following identities hold:*

$$\begin{aligned} & \left\{ \sum_{i=1}^{l} (c_i p)^2 \mid l \in \mathbb{N}, c_i \in \mathbb{Q}^k \right\} \\ = & \left\{ \sum_{i=1}^{l} \alpha_i (c_i p)^2 \mid l \in \mathbb{N}, \alpha_i \in \mathbb{Q}, \alpha_i \geq 0, c_i \in \mathbb{Q}^k \right\} \\ = & \left\{ p^t X p \mid X \in \mathbb{Q}^{k \times k} \text{ positive semidefinite} \right\} \end{aligned}$$

Proof. The first equation holds because each non-negative rational number α_i can be written as a sum of four rational squares by Lagrange's four-square theorem.

We consider the two directions of the second equation:

"$\supseteq$": This is shown (constructively) by [Har07, Theorem 1].

"$\subseteq$": Let $\sum_{i=1}^{l} \alpha_i (c_i p)^2$ be a sum of squares with $\alpha_i \geq 0$. We define the matrices

$$C = \begin{pmatrix} c_1^t \\ c_2^t \\ \vdots \\ c_l^t \end{pmatrix}, \qquad D = \begin{pmatrix} \alpha_1 & & & \\ & \alpha_2 & & \\ & & \ddots & \\ & & & \alpha_l \end{pmatrix}$$

and thus obtain the identity:

$$\sum_{i=1}^{l} \alpha_i (c_i p)^2 \;=\; (Cp)^t D (Cp) \;=\; p^t (C^t D C) p$$

Let $X \in \mathbb{Q}^{k\times k}$ be a matrix such that $X = C^t DC$. This matrix is positive semidefinite because of:

$$x^t(C^tDC)x \;=\; \sum_{i=1}^{l} \alpha_i (c_i x)^2 \;\geq\; 0 \;. \qquad \square$$

By combining Lemma 15 with the NSS, we see that a set G of polynomials does not have any common zeroes if and only if there is a vector p of polynomials and a PSD matrix $X \in \mathbb{R}^{k\times k}$ such that $1 + p^t Xp \in (G)$. As the vector space of polynomials is generated by monomials, it is sufficient to consider vectors p of monomials.

Semidefinite programming [BV04] provides a simple method to determine such matrices X. A *semidefinite program* (SDP) is an optimization problem in terms of traces (tr) of matrices:

maximise	$\mathrm{tr}(CX)$
subject to	$\mathrm{tr}(A_i X) = b_i \quad (\text{for } i \in \{1,\ldots,n\})$,
where	X positive semidefinite

where $A_i, C \in \mathbb{R}^{k\times k}$ are symmetric matrices and $b_i \in \mathbb{R}$. Such optimization problems can be solved efficiently using numerical convex optimization [BV04].

The key insight underlying this method is the following: by computing a Gröbner basis B for the ideal (G), the NSS condition $1 + p^t Xp \in (G)$ can be encoded as the linear side constraints $\mathrm{tr}(A_i X) = b_i$ $(i \in \{1,\ldots,n\})$ of a semidefinite program searching for X. To see this, note that both the expression $1 + p^t Xp$ and the reduction $\mathrm{red}_B(1 + p^t Xp)$ are linear in X. Because Gröbner bases determine unique remainders, we therefore have $1 + p^t Xp \in (G)$ if and only if $\mathrm{red}_B(1 + p^t Xp) = 0$. This equation is a linear constraint on X suitable for SDP.

To capture this observation formally, let Q be a symmetric $k \times k$ matrix of parameters:

$$Q \;=\; \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,k} \\ q_{1,2} & q_{2,2} & \cdots & q_{2,k} \\ \cdots & \cdots & \cdots & \cdots \\ q_{1,k} & q_{2,k} & \cdots & q_{k,k} \end{pmatrix}$$

The polynomial $1 + p^t Qp$ is linear in Q and can be represented in the form

$$1 + p^t Qp = q^t C m \;,$$

where $q = (q_{1,1}, q_{1,2}, \ldots, q_{k,k})^t$ is the vector of all the Q-parameters,

$$m = (m_1, \ldots, m_s)^t$$

is a vector of monomials over $X_1, \ldots, X_n$ (containing, at least, 1 and all products $p_i p_j$ of components of p), and $C \in \mathbb{Q}^{k^2 \times s}$ is a matrix. By computing the remainder $q^t Dm = \mathrm{red}_B(q^t Cm)$ of this term for a Gröbner basis B over $\mathbb{Q}[X_1, \ldots, X_n]$, we can construct the required side constraints:

Lemma 16. *Suppose that the components of m are pairwise distinct, and that $q^t Cm, q^t Dm \in \mathbb{Q}[q_{1,1}, q_{1,2}, \ldots, q_{k,k}][X_1, \ldots, X_n]$ are two polynomials defined by the matrices $C, D \in \mathbb{Q}^{k^2 \times s}$, such that $q^t Dm = \mathrm{red}_B(q^t Cm)$. Then the following equation holds:*

$$\{x \in \mathbb{R}^k \mid red_B(x^t Cm) = 0\} \;=\; \{x \in \mathbb{R}^k \mid x^t D = 0\} \tag{6.2}$$

Proof. First, observe that for all $x \in \mathbb{R}^n$:

$$x^t Cm - x^t Dm \in (B) \tag{6.3}$$

The proof of the lemma is as follows:

- "$\supseteq$": Suppose $x^t D = 0$. Then also $x^t Dm = 0$ and, by (6.3),

  $$x^t Cm - x^t Dm = x^t Cm \in (B) \ .$$

  This implies $\mathrm{red}_B(x^t Cm) = 0$ because B is a Gröbner basis.

- "$\subseteq$": Suppose $\mathrm{red}_B(x^t Cm) = 0$, i.e., $x^t Cm \in (B)$. By (6.3), this implies $x^t Dm \in (B)$.

  Now, observe that also the instance $x^t Dm$ is irreducible w.r.t. B: because the parametrized polynomial $q^t Dm$ is irreducible w.r.t. B, it has to be the case that the i'th component of $b^t D$ is zero whenever the monomial m_i is reducible w.r.t. B. This means that, in this case, the i'th column of D only contains zeroes. Then also the i'th component of $x^t D$ is zero and $x^t Dm$ cannot contain any reducible terms.

  Because B is a Gröbner basis, 0 is the only member of (B) that is irreducible with respect to B, which implies $x^t Dm = 0$. Finally, because the elements of m are pairwise distinct and thus linearly independent, this is only possible if $x^t D = 0$. □

Example 26. *We return to the implication* (6.1) *proven in Figure 6.5 by showing that the polynomials* $B = \{a^2 - x + y,\ b^2 - z,\ xzc^2 - yzc^2 + 1\}$ *have no common zeroes. The witness* $1 + (abc)^2$ *used in the proof of Figure 6.5 can be constructed systematically for a suitable set of basis monomials, say,* $p = (1, a^2, abc)^t$*. We need to find a PSD matrix* $X \in \mathbb{Q}^{3\times 3}$ *such that* $1 + p^t X p \in (B)$*. To do so, we compute the reduction* $\mathrm{red}_B(1 + p^t Q p)$ *for a symbolic* 3×3 *parameter matrix* Q*:*

$$\begin{aligned}&\mathrm{red}_B(1 + p^t Q p)\\ &= \mathrm{red}_B(1 + q_{1,1}1^2 + 2q_{1,2}a^2 + 2q_{1,3}abc + 2q_{2,3}a^3bc + q_{3,3}a^2b^2c^2)\\ &= 1 + q_{1,1} - q_{3,3} + 2q_{1,2}x - 2q_{1,2}y + 2q_{1,3}abc + 2q_{2,3}abcx - 2q_{2,3}abcy\end{aligned}$$

By comparing coefficients, the constraints on Q *for this polynomial to be* 0 *are:*

$$\begin{aligned} 1 + q_{1,1} - q_{3,3} &= 0 & -2q_{1,2} &= 0 & 2q_{2,3} &= 0\\ 2q_{1,2} &= 0 & 2q_{1,3} &= 0 & -2q_{2,3} &= 0\end{aligned}$$

A positive semidefinite solution of the constraints is $q_{3,3} = 1$ *and* $q_{i,j} = 0$ *for all* $(i,j) \neq (3,3)$*, which means* $1 + p^t Q p = 1 + (abc)^2$*.*

Theorem 14 (Completeness [PQR09a])**.** *By enumerating all monomials for* p *successively, Gröbner bases for the real Nullstellensatz give a complete method for universal real arithmetic: If the original formula is valid, then, when* p *contains all monomials of a sufficiently large degree, the corresponding semidefinite programs will have a solution (the witness).*

Proof. The proof is a combination of Lemma 16 with Corollary 8. □

6.4.2.1 Discussion and Practical Considerations

Semidefinite programming turns the search for witnesses $1 + s_1^2 + \cdots + s_m^2$ into a (simpler) search for suitable basis monomials p. As the number of basis monomials that need to be considered is finite (due to degree bounds on witnesses [Par03]), this yields a theoretical decision procedure. Practically, we enumerate all monomials with ascending degree. There might be more sophisticated methods, however: the number of monomials that witnesses are actually built of is usually small, and it might be possible to locate likely candidates by analyzing the Gröbner basis B. In our experience, the number of basis monomials that are considered before a solution

is found (and thus the difficulty of a problem) depends on (i) the number of variables in the polynomial ring, and (ii) the degree of the leading monomials in the Gröbner basis.

Another issue is that implementations for semidefinite programming (like the CSDP solver [Bor99] used by us) are numerical and produce answers in floating point arithmetic. To recover precise solutions in $\mathbb{Q}$ from such answers, we use a similar approach as in [Har07]: We approximate floating point numbers to a certain precision by rationals, and check resulting solution candidate for semidefiniteness. In order to get rational candidates for the floating point numbers we use Stern-Brocot trees [GKP94]. These trees enumerate all fractions $\frac{a}{b}$ such that their greatest common divisor is 1, i.e., $gcd(a, b) = 1$. For a floating point number x we start the tree construction with the "fractions" $\frac{\lfloor x \rfloor}{1}$ and $\frac{1}{0}$. The construction rule for Stern-Brocot trees now creates a new fraction as the sum of denominators divided by the sum of the numerators. We call the two inputs the left and the right fraction. If the resulting fraction is smaller than x then we replace the left fraction, if it is greater than x we replace the right one. Then we reiterate the procedure. Once we get a fraction that with sufficient precision approximates x we perform the same procedure to get approximations for the other floating point numbers in the output of CSDP and check for semidefiniteness of the matrix. If the matrix is semidefinite we are done, if not we increase the precision successively as long as the solution candidate remains indefinite.

Optimizations. We found it essential to use preprocessing steps to reduce the number of variables in a problem, such that the number of potential basis monomials becomes tractable. Some heuristics are:

- If the Gröbner basis B contains a polynomial $x + t$ such that x does not occur in t, then x and the polynomial can be eliminated by simple rewriting.
- If B contains polynomials $xy - 1$ and $x^n + t$ such that x^n does not divide t, then x and the polynomial $xy - 1$ can be eliminated by multiplying each polynomial in B (except $xy - 1$) with a power of y and reducing w.r.t. $xy - 1$.
- Polynomials $\alpha_1 m_1^2 + \cdots + \alpha_n m_n^2 \in B$ for which all coefficients α_i with $i \in \{1, \ldots, n\}$ are strictly positive can be replaced by the monomials $m_1, \ldots, m_n$.

- If B contains a polynomial $\alpha_0 x^2 - \alpha_1 m_1^2 - \cdots - \alpha_n m_n^2$ such that $\alpha_i > 0$ for $i \in \{0, \ldots, n\}$ where x only occurs with even degree in B, then x can be eliminated by rewriting and the polynomial can be removed.

The last two cases are surprisingly common, due to the encoding of inequalities by quadratic terms performed by A4 and A5.

6.4.3 Experimental Results

We have integrated the techniques presented in Section 6.4.1–6.4.2 into KeYmaera. With the various methods for real arithmetic integrated into a common framework and real arithmetic examples from different domains, we have a solid base for our experiments. The benchmarks are a collection of challenging arithmetic problems from the hybrid system world [PQ08b, PQ09a, Pla10b, LPN11, ALPK12, MLP12], the verification of invariant properties for mathematical algorithms [Kov08, dMB08] and algebraic geometry [DSW98], as well as a smaller number of synthetic problems. For the examples with mixed quantifiers, our setting applies QM to the existential quantifiers such that we can still gain insight into the scalability of the approaches that are restricted to the universal fragment on these examples. We run our experiments on a dual Intel Xeon E5430 (quad core with 2.66 GHz) and 32 gigabytes RAM.

The experimental results are summarized in Figure 6.8 and Tabular 6.1. We have used a timeout of 200 seconds on the 1344 examples. The results show that, for our particular mix of examples, quantifier elimination procedures are still faster than recent approaches with semidefinite programming relaxations for the Positivstellensatz, while Gröbner bases alone have difficulties with "real" problems. As expected, procedures tailored for real arithmetic can solve substantially more cases than Gröbner bases for general fields. Gröbner bases that integrate Fourier-Motzkin (GK) solve many more problems. Z3 as the newest approach in our tool set performs pleasingly well. For many easy examples it is a lot faster than QM and solves almost as many examples. Interestingly, QR and QR_q perform almost identical while QQ_r performs better than QQ. Thus, for our set of examples the simplifications performed by Redlog are a lot more important than the implementation of CAD.

Our combination, GRN, of Gröbner bases with the real Nullstellensatz is competitive with quantifier elimination by partial cylindrical algebraic

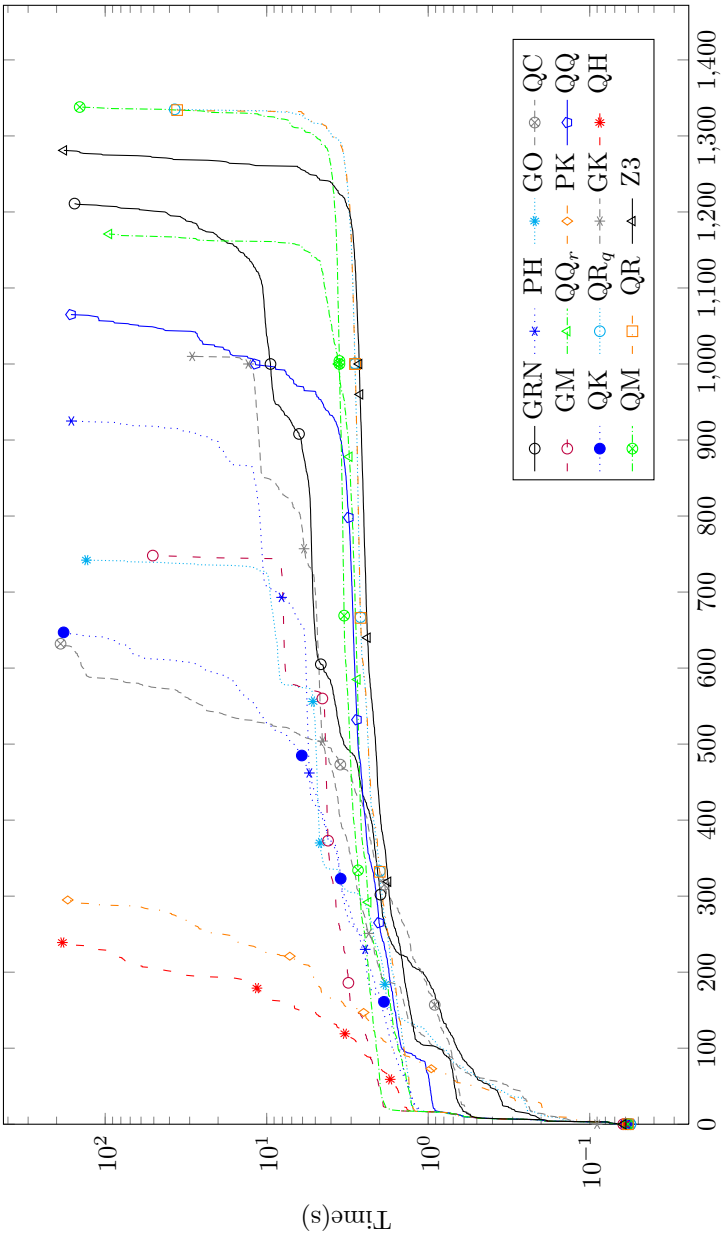


Figure 6.8: Examples solved per time

Table 6.1: Individual examples solved within x seconds

Conf.	1s	5s	10s	30s	60s	90s	120s	180s	200s
GO	120	519	725	738	740	742	742	743	743
GM	17	570	745	747	749	749	749	749	749
GK	112	700	850	1011	1011	1011	1011	1011	1011
PK	75	190	226	266	286	288	291	296	296
PH	17	420	706	911	919	924	924	926	926
GRN	189	640	1037	1188	1203	1208	1210	1212	1212
Z3	104	1249	1261	1269	1274	1275	1277	1281	1282
QH	17	148	167	196	209	225	235	239	240
QC	177	506	527	559	579	586	588	630	633
QK	17	437	528	608	615	637	642	647	648
QQ	66	964	998	1044	1050	1057	1064	1066	1066
QQ_r	17	1140	1162	1166	1169	1171	1172	1172	1172
QR	17	1318	1333	1334	1335	1335	1335	1335	1335
QR_q	17	1317	1334	1335	1336	1336	1336	1336	1336
QM	17	1300	1326	1334	1336	1336	1338	1339	1339

decomposition [CH91] and finishes as the 5th best approach behind QR, QR_q, QM, and Z3. The experiments also show that substantial performance improvements (QR and QM) are still possible beyond partial CAD.

The experiments show that GM and GO are on a par with a slight edge for GM. Further, QM, QR, QR_q are very close, but clearly outperform QQ_r, QQ, QC, and QK both in runtime and number of provable cases. The two implementations of Cohen-Hörmanders procedure QC and QK are on par w.r.t. the number of examples solvable though in terms of runtime QC is faster on many examples. QH is slower but competitive with the number of examples solved by PK but does not yet perform as well as other QE implementations, Z3, or GRN. The performance gap between PK and PH is surprising. In part, it shows how important Harrison's optimizations [Har07] of Parrilo's work [Par03] are, but may also be caused by different heuristics for recovering rationals from floats and different enumeration orders for polynomials. This might indicate that PK, indeed, gives a more objective comparison for GRN than PH, because PK and GRN share exactly the same KeYmaera framework and rational recovering. Our GRN procedure is a clear progress compared to PK. Inevitably, performance depends on the system options and on the set of benchmarks.

6.4.4 Related Work

Nipkow [Nip08] presented a formally verified implementations of quantifier elimination in an executable fragment of Isabelle/HOL, currently for linear real arithmetic only. McLaughlin and Harrison [MH05] presented a nonverified but proof-producing implementation of general quantifier elimination, so that the result of the procedure can be checked independently.

The sum of squares approach has been pioneered by Parrilo [Par03] and Harrison [Har07]. Harrison also gives optimizations for the univariate case.

Tiwari [Tiw05] presents an approach using Gröbner bases and sign conditions on variables to produce unsatisfiability witnesses for nonlinear constraints. The approach depends on appropriate heuristic variable orderings that are formed by successively introducing new variables for polynomial expressions following certain heuristics (which may not terminate). Our work and that of Tiwari share the combination of Gröbner bases with witness generation. Yet we follow semi-definite programming for the real Nullstellensatz, whereas [Tiw05] uses heuristic generation of polynomial witness expressions. Tiwari uses the Positivstellensatz to prove refutational completeness but not as part of his technique.

RSolver [Rat06] is a numerical approach for deciding validity of (robust instances of) first-order formulas over real arithmetic extended with transcendental functions. Unlike our work, this relies on numerical stability of the input formula.

MetiTarski [AP07] is an interesting approach for handling special functions using a combination of resolution proving with simple QE procedures. Their focus is on handling special functions not on handling real arithmetic. Recently, Andrew Sogokon implemented an interface connecting MetiTarski and KeYmaera thus making it available as a back-end when proving properties of hybrid systems and hybrid games. We decided to not include it in our comparison in this chapter as the focus deviates from that of the other approaches. In the current benchmark set it would just apply its fall-back procedures like Mathematica's quantifier elimination procedure and thus produce almost identical results.

Hunt et al. [WAHKM03] describe the handling of nonlinear arithmetic in ACL2, which is based on heuristic multiplication of inequalities in the style of (6.1) and yields an incomplete method. The method is claimed to be empirically successful, though, and can also be applied to nonlinear integer arithmetic.

6.4.5 Discussion and Conclusions

The respective approaches from Section 6.4.1–6.4.2 have different advantages and weaknesses for formal verification of real world problems in real arithmetic. We draw a qualitative comparison complementing the quantitative comparison from Section 6.4.3.

Quantifier Elimination. Quantifier elimination procedures [CH91] can handle full nonlinear real arithmetic, including existential quantifiers. Their implementations are quite intricate algorithms for which correctness is not easily established formally. Unfortunately, QE does not produce simple checkable certificates.

Proof-producing [MH05] or verified [Nip08] QE procedures may be interesting improvements on the formal traceability of QE w.r.t. to generating checkable proofs or certificates. Unfortunately, their performance is not yet fully competitive with other quantifier elimination implementations, SMT based approaches like Z3, or our proof-producing GRN procedure.

A compromise is reverification: Proof search [PC08,Pla07a] in KeYmaera generates several problems of real arithmetic to find a proof, but only those in the final proof are soundness-critical. For soundness, it is sufficient to use a fast or untrusted implementation of QE during the proof search and to reverify the final proof in a proof checker with a verified or proof-producing QE implementation [MH05,Nip08]. For this purpose, KeYmaera strategies are especially useful that identify the sweetspot for applying QE iteratively during the proof search [Pla07a].

Positivstellensatz. In the context of verification, a useful property of the Positivstellensatz is that it produces a witness ($s + g + m^2 = 0$) for the validity of a formula. Once the witness has been found, it is checkable by simple computations in the polynomial ring to determine whether the polynomial identity holds by comparing the coefficients. Similarly, the well-formedness of the witness can be determined by checking whether s is build from sums of squares using an extension of "completing the square" [Har07]. Thus, complicated numerical semidefinite programming tools [BV04] do not need to be part of the trusted computing base concerning soundness. Due to its enumerative nature with a large number of extra parameters, scalability with the number of variables is still limited.

Gröbner Bases. The Gröbner Basis approach does not have simple witnesses like Positivstellensatz approaches. Their working principle, however, is strictly based on symbolic computations, which can be carried out from a small set of rewrite rules within a logic. This corresponds to our built-in Gröbner basis approach GK, which is almost as efficient as external Gröbner basis implementations. Our experimental results indicate that, due to the partial ignorance of real-closed field properties, the capabilities of Gröbner bases alone are not sufficient, even in combination with Fourier-Motzkin elimination. Still, GK is able to outperform other approaches like the Cohen-Hörmander procedures QC and QK, and even be competitive with CAD based methods like QQ on a large set of benchmarks.

SMT for real-closed fields. The recent advances in SAT modulo theory resulted in an approach for dealing with real arithmetic using conflict-driven clause learning. From our experiments we can see that the implementation of these ideas in Z3 performs faster than most other methods. Only virtual substitution implemented in Redlog (QR and QR_q) and the commercial tool Mathematica (QM) can solve more examples.

Real Nullstellensatz. Our decision procedure based on Gröbner basis computations and the real Nullstellensatz share the presence of checkable witnesses with approaches based on the Positivstellensatz. Once a witness $1 + \sum_i s_i^2 = 0$ has been found, the polynomial equality check can be performed easily within a proof system using the GK rules, giving a fully formal proof. The performance in our experiments show that this approach is promising. It outperforms most other approaches, except for highly tuned QE procedures, which lack support for formal traceability and the new competitor Z3 which performs extremely well while producing checkable proofs. We believe that further research in this area is likely to produce competitive but traceable solutions for real arithmetic.

CHAPTER SEVEN

Case Study

For a moment, nothing happened. Then, after a second or so, nothing continued to happen.
— Douglas Adams

Contents

In this chapter we consider a case study taken from the domain of train control. That is, we analyze a model of the *European Train Control System* (ETCS) [ERT02]. The ETCS is an approach to unify train control and train protection units all over Europe while establishing additional goals like maximzing throughput in order to be able to transport millions of passengers throughout Europe each day.

Related Work. Train control and especially the ETCS has been used as a case study by different authors in the past. We list some literature about the application of formal methods in the domain of train control. Peleska et al. [PGHD04] verify routing algorithms for trains using bounded model checking. Faber et al. [FIJSS10] consider real-time aspects of a dynamically reconfigurable model of the ETCS. They model the system in UML using the graphical modeling tool Syspect [FLOQ11]. The semantics of this annotated UML model is defined in terms of CSP-OZ-DC, a combination of the specification languages communicating sequential processes (CSP) [Hoa78] to model the operational behavior of systems, Object-Z (OZ) [Smi00] to model infinite data types, and the duration calculus (DC) [ZHR91] to specify real-time properties. On this semantics, Faber et al. show safety properties of the train system by exploiting the fact that the theory of the list structures used in this model form a local theory extension [SS05]. Thus, they can reduce reasoning about list structures in this case to a decidable base theory. Cimatti et al. [CRT09] consider the consistency of informal requirements on ETCS. These requirements are expressed as temporal properties including the continuous system dynamics. They analyze the requirements using an approach based on the combination of temporal logic with regular expressions. Banach et al. [BZSH12] use a train model in order to demonstrate their retrenchment based approach to control design. We [QS06] formally verify correct functioning of a gate controller modeled in the Shape Calculus [Sch06], a multi-dimensional extension of the Duration Calculus. By translating the Shape Calculus specification into monadic second-order formulas and applying the model checker MONA [EKM98] we fully automatically verfiy spatial and real-time properties of the controller. In [PQ08b, PQ09a] we have studied different properties of a parametric version of the ETCS. We used our theorem prover to show that our model is controllable, reactive, safe, as well as live.

Contributions. We present a small case study from the setting of train control. We prove that our specification is safe in the sense that the train does not exceed the end of its movement authority and show for a family of implementations with different communication delays that they are robust refinements of our specification. Thus, we can conclude that these are safe as well.

Structure of this Chapter. In Section 7.1 we give a general introduction in the relevant parts of the ETCS for our case study. These results have been partially published in [PQ08b, PQ09a, PQ09b]. We provide a model of a train controller and RBC in Section 7.2. Subsequently, we provide a family of implementations that suffer from communication delays and show that these are robust refinements of our original specification in Section 7.3. We discuss the results of this chapter in Section 7.4.

7.1 Overview

The *European Train Control System* (ETCS) [ERT02] has a wide range of different possible configurations of trains, track layouts, and different driving circumstances. It is a standard to ensure safe and collision-free operation as well as high throughput of trains at speeds up to 320km/h. ETCS level 3 follows the *moving block principle*, i.e., movement permissions are neither known beforehand nor fixed statically. They are determined based on the current track situation by a *Radio Block Controller* (RBC). Trains are only allowed to move within their current *movement authority* (MA), which can be updated by the RBC using wireless communication. Hence the train controller needs to regulate the movement of a train locally such that it always remains within its MA. Behind MA, there could be open gates, other trains, or speed restrictions due to tunnels. The automatic train protection unit (*atp*) dynamically determines a safety envelope around a train, within which it considers driving safe, and adjusts the train acceleration a accordingly. Figure 7.1 illustrates the dynamic assignment of MA. The ETCS controller switches according to the protocol pattern in Figure 7.2 which corresponds to a simplified version of the protocoal studied by Damm et al. [DMO$^+$07]. When approaching the end of its MA the train switches from *far* mode (where speed can be regulated freely) to negotiation (*neg*), which, at the latest, happens at the point indicated by *ST* (for *start talking*). During negotiation the RBC grants or denies MA-extensions. If the extension is not granted in time, the train starts braking in the correcting mode (*cor*) returning to *far* afterwards. Emergency messages announced by the RBC can also put the controller into *cor* mode. If so, the train switches to a failsafe state (*fsa*) after the train has come to a full stop and awaits manual clearance by the train operator.

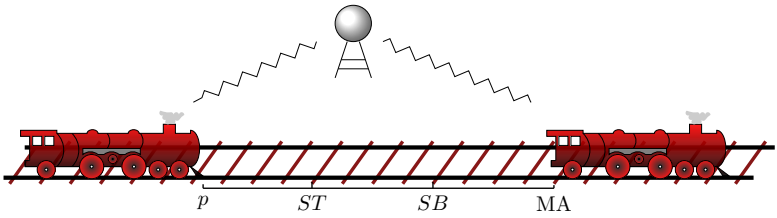


Figure 7.1: ETCS train cooperation protocol (Dynamic assignment of movement authorities)

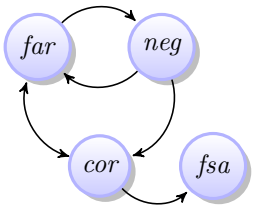


Figure 7.2: ETCS train cooperation protocol (Cooperation pattern)

Lemma 17 (Principle of separation by movement authorities)**.** *If each train stays within its MA and, at any time, MAs issued by the RBC form a disjoint partitioning of the track, then trains can never collide (proof see [PQ09b]).*

Lemma 17 effectively reduces the verification of an unbounded number of traffic agents to a finite number. We exploit MAs to decouple reasoning about global collision freedom to local cooperation of every traffic agent with the RBC. In particular, we verify correct coordination for a train without having to consider gates or railway switches, because these only communicate via RBC mediation and can be considered as special reasons for denial of MA-extensions. We only need to prove that the RBC handles all interaction between the trains by assigning or revoking MA correctly and that the trains respect their MA. However, to enable the RBC to guarantee disjoint partitioning of the track it has to rely on properties like appropriate safe rear end computation of the train. Additionally, safe operation of the train plant in conjunction with its environment depends

on proper functioning of the gates. As these properties have a more static nature, they are much easier to show once the actual hybrid train dynamics and movements have been proven to be controlled correctly.

As trains are not allowed to drive backwards without clearance by track supervision personnel, the relevant part of the safety envelope is the closest distance to the end of its current MA. The point SB, for *start braking*, is the latest point where the train needs to start correcting its acceleration (in mode *cor*) to make sure it always stays within the bounds of its MA.

7.2 Specification

We consider an event driven train model. Whenever the distance to the end of the movement authority is large enough, in that case the train accelerates with maximal acceleration A. If the train approaches the end of the movement authority it eventually switches into a mode where it applies maximal braking force. A first version of our model is depicted in Figure 7.3. The state of the train is stored in the variables p (position), v (velocity), and a (acceleration). The RBC model consists of a clock c to measure its cycle time and a variable MA that stores the current end of the movement authority. This variable is assumed to be shared with the train. We assume that MA is updated as the message saying that the movement authority has been extended is received by the train. In line 1 a message from the RBC is received and the internal representation of the movement authority MA is updated. We assume that this happens every n-time units and that the extension is of length $mp > 0$. This time is measured by the clock c. Line 2 models the train controller. Whenever the distance to the end of the movement authority is greater than the absolute braking distance, the train chooses the maximum acceleration A. Otherwise, it chooses the maximum deceleration $-b$. The train dynamics is split into two modes (line 4 and 6) in order to make sure that braking does not lead to a negative velocity. That is, in case the veolocity is positive, the train follows the acceleration chosen by its controller (line 4). However, the evolution is restricted by $v \geq 0$ in order to make sure that the train is not driving backwards. When the velocity is 0, then the maximal deceleration is 0 (modeled by the $\max\{a, 0\}$ in line 6).

We assume that the system is started in an initial region where all system variables are 0 and the parameters A and b are positive numbers. Using the invariant $v^2 \leq 2b(\mathrm{MA}-p) \wedge v \geq 0$ we can prove that the system always stays

1 $\big($if $c \geq n$ then MA := MA $+ mp$; $c := 0$ fi;
2 if $v^2 < 2b(\text{MA} - p)$ then $a := A$ else $a := -b$ fi;
3 if $v > 0$ then
4 $(\dot{p} = v, \dot{v} = a, \dot{c} = 1, v^2 \leq 2b(\text{MA} - p), v \geq 0, c \leq n)$
5 else
6 $(\dot{p} = v, \dot{v} = \max\{a, 0\}, \dot{c} = 1, v^2 \leq 2b(\text{MA} - p), c \leq n)$
7 fi$\big)^*$

Figure 7.3: ETCS model

within its movement authority, i.e., $\Box(p \leq \text{MA})$, where $\Box F \mathrel{\widehat{=}} \Box_{[0,\infty[} F$. This can be done fully automatically using our theorem prover KeYmaera.

7.3 Robust Refinements

When the initial region satisfies $c = 0$ then there is no communication delay. In case $c = -com$ for $com > 0$ holds in the initial region, then there is a communction delay of exactly com time units. Let $com = 0$ hold for our specification S. We conjecture that for an implementation I_{com} with some value for $com \geq 0$, I_{com} is an com-0-refinement of the specification, i.e., $I_{com} \leadsto^{com,0} S$. In order to prove this, we first transform our program into standard form. The result is depicted in Figure 7.4. It can be obtained by applying the definition of the if-statement and the distributivity law of sequential composition and nondeterministic choice, i.e.,

$$\alpha; (\beta \cup \gamma) \equiv (\alpha; \beta) \cup (\alpha; \gamma) \ .$$

Observe that, in line 8 the choice of the acceleration is 0 as opposed to $-b$ in line 3 in the case of braking. This way, we could avoid the max-function in the differential equation systems in line 4 and 9. We visualize the trajectories for the case that there is no communication delay and an implementation with $com = 2$. We use the following parameter instances for the simulation in Figure 7.5 and Figure 7.6:

$$n = 7 \wedge mp = 10 \wedge A = 1 \wedge b = 2$$

$$
\begin{array}{l}
\Big(\big(?v > 0; \\
\quad \text{if } c \geq n \text{ then } \mathrm{MA} := \mathrm{MA} + mp;\ c := 0 \text{ fi}; \\
\quad \text{if } v^2 < 2b(\mathrm{MA} - p) \text{ then } a := A \text{ else } a := -b \text{ fi}; \\
\quad (\dot{p} = v, \dot{v} = a, \dot{c} = 1, v^2 \leq 2b(\mathrm{MA} - p), v \geq 0, c \leq n)\big) \\
\cup \\
\big(?v \leq 0; \\
\quad \text{if } c \geq n \text{ then } \mathrm{MA} := \mathrm{MA} + mp;\ c := 0 \text{ fi}; \\
\quad \text{if } v^2 < 2b(\mathrm{MA} - p) \text{ then } a := A \text{ else } a := 0 \text{ fi}; \\
\quad (\dot{p} = v, \dot{v} = a, \dot{c} = 1, v^2 \leq 2b(\mathrm{MA} - p), c \leq n)\big) \\
\Big)^*
\end{array}
$$

Figure 7.4: ETCS model in standard form

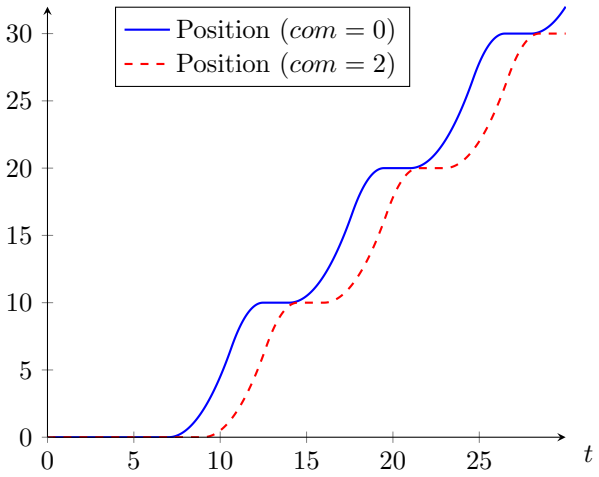


Figure 7.5: Position of a train with and without message delay

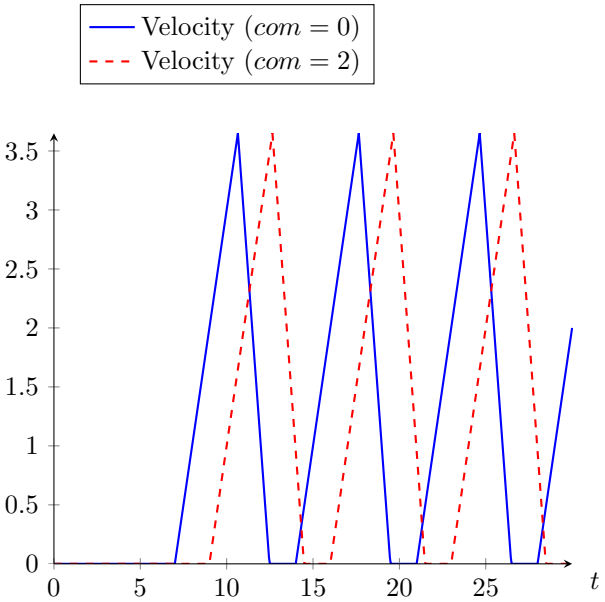


Figure 7.6: Velocity of a train with and without message delay

Using the model in standard form, we can construct a game as described in Section 5.2. The resulting game, without the initialization phase is given in Figure 7.7. Falsifier controls the implementation I_{com} with variables index by 1. Verifier controls the specification S with variables index by 2. In order to simplify the proof search we decided to encode the control strategy for the retiming parameter s directly into the game thereby restricting the possible strategies of Verifier. The game consists of a choice for Falsifier whether he is in the mode where $v_1 > 0$ (lines 1-16), the mode where $v_1 = 0$ (line 17-32), or wants to take no action at all (line 33). Note that his choice whether $v_1 > 0$ or $v_1 = 0$ is determined by the current value of v_1. If he chooses one of his regular modes, there is the same choice for Verifier determined by the value of v_2. Verifier chooses the value of s as follows: In case the velocity is positive he chooses $s = 1$ (lines 7 and 23). Otherwise, he chooses $s = 2$ as long as this is possible, i.e., as long as $r < \delta$ and $s = 1$ if $r = \delta$ (lines 14 and 30). At the end of each loop iteration Verifier has to ensure that the distance between of the positions, velocities, and movement authorities is 0 (line 33).

In order to prove that Verifier still has a winning strategy in this game, we need to find an inductive loop invariant. As a minimum requirement this loop invariant has to imply that the distance between the state vectors of the implementation and the specification is 0. In addition it needs to encode sufficient knowledge about the value of r during the execution of the loop in order to allow for reasoning that the strategy respects the temporal bounds, i.e., $|r| \leq \varepsilon$. We prove that the following formula is invariant for the loop in Figure 7.7 using KeYmaera:

$$\begin{aligned} &x_1 = x_2 \wedge v_1 = v_2 \wedge m_1 = m_2 \\ &\quad \big((-\varepsilon \leq c_1 \leq 0 \wedge r - c_1 = \varepsilon \wedge c_2 = 0 \wedge x_1 = 0 \wedge v_1 = 0 \\ &\qquad \wedge m_1 = 0 \wedge 0 \leq r \leq \varepsilon) \vee (c_1 = c_2 \wedge r = \varepsilon)\big) \end{aligned}$$

The invariant encodes that with an increasing value of r towards 2 the distance between c_1 and c_2 converges to 0. Further, while the values of c_1 and c_2 are not yet equal neither of the trains moves. That is, even though this whole retiming process might consume more than n time units, no time passes in our specification as Verifier chooses $s = 2$. We use the following assumptions about our system parameters for the proof:

$$A > 0 \wedge b > 0 \wedge n > 0 \wedge mp > 0 \ .$$

$$
\begin{array}{l}
\Big(\Big(\Big(\Big[(?v_1 > 0; \\
\quad \text{if } c_1 \geq n \text{ then } \mathrm{MA}_1 := \mathrm{MA}_1 + mp;\ c_1 := 0 \text{ fi}; \\
\quad \text{if } v_1^2 < 2b(\mathrm{MA}_1 - p_1) \text{ then } a_1 := A \text{ else } a_1 := -b \text{ fi}\Big] \\
\quad \Big(\big(\langle ?v_2 > 0; \\
\quad \text{if } c_2 \geq n \text{ then } \mathrm{MA}_2 := \mathrm{MA}_2 + mp;\ c_2 := 0 \text{ fi}; \\
\quad \text{if } v_2^2 < 2b(\mathrm{MA}_2 - p_2) \text{ then } a_2 := A \text{ else } a_2 := -b \text{ fi}\big\rangle \\
\quad \langle s := 1\rangle \\
\quad [\dot{p}_1 = s \cdot v_1, \dot{v}_1 = s \cdot a, \dot{c}_1 = s, v_1^2 \leq 2b(\mathrm{MA}_1 - p_1), v_1 \geq 0, c_1 \leq n, \\
\quad \dot{p}_2 = (2-s)v_2, \dot{v}_2 = (2-s)a, \dot{c}_2 = 2-s, v_2^2 \leq 2b(\mathrm{MA}_2 - p_2), v_2 \geq 0, \\
\quad c_2 \leq n, \dot{r} = 2s - 2, |r| \leq \varepsilon]\Big) \\
\quad \cup \\
\quad \big(\langle ?v_2 \leq 0; \\
\quad \text{if } c_2 \geq n \text{ then } \mathrm{MA}_2 := \mathrm{MA}_2 + mp;\ c_2 := 0 \text{ fi}; \\
\quad \text{if } v_2^2 < 2b(\mathrm{MA}_2 - p_2) \text{ then } a_2 := A \text{ else } a_2 := 0 \text{ fi}\big\rangle \\
\quad \langle \text{if } r < \varepsilon \text{ then } s := 2 \text{ else } s := 1 \text{ fi}\rangle \\
\quad [\dot{p}_1 = s \cdot v_1, \dot{v}_1 = s \cdot a, \dot{c}_1 = s, v_1^2 \leq 2b(\mathrm{MA}_1 - p_1), v_1 \geq 0, c_2 \leq n, \\
\quad \dot{p}_2 = (2-s)v_2, \dot{v}_2 = (2-s)a, \dot{c}_2 = 2-s, v_2^2 \leq 2b(\mathrm{MA}_2 - p_2) \\
\quad c_2 \leq n \wedge \dot{r} = 2s - 2, |r| \leq \varepsilon]\Big)\Big)\Big) \\
\cap \Big(\Big[(?v_1 \leq 0; \\
\quad \text{if } c_1 \geq n \text{ then } \mathrm{MA}_1 := \mathrm{MA}_1 + mp;\ c_1 := 0 \text{ fi}; \\
\quad \text{if } v_1^2 < 2b(\mathrm{MA}_1 - p_1) \text{ then } a_1 := A \text{ else } a_1 := 0 \text{ fi}\Big] \\
\quad \Big(\big(\langle ?v_2 > 0; \\
\quad\quad \text{if } c_2 \geq n \text{ then } \mathrm{MA}_2 := \mathrm{MA}_2 + mp;\ c_2 := 0 \text{ fi}; \\
\quad\quad \text{if } v_2^2 < 2b(\mathrm{MA}_2 - p_2) \text{ then } a_2 := A \text{ else } a_2 := -b \text{ fi}\big\rangle \\
\quad\quad \langle s := 1\rangle \\
\quad\quad [\dot{p}_1 = s \cdot v_1, \dot{v}_1 = s \cdot a, \dot{c}_1 = s, v_1^2 \leq 2b(\mathrm{MA}_1 - p_1), c_1 \leq n, \\
\quad\quad \dot{p}_2 = (2-s)v_2, \dot{v}_2 = (2-s)a, \dot{c}_2 = 2-s, v_2^2 \leq 2b(\mathrm{MA}_2 - p_2), v_2 \geq 0, \\
\quad\quad c_2 \leq n, \dot{r} = 2s - 2, |r| \leq \varepsilon]\Big) \\
\quad \cup \\
\quad\quad \big(\langle ?v_2 \leq 0; \\
\quad\quad \text{if } c_2 \geq n \text{ then } \mathrm{MA}_2 := \mathrm{MA}_2 + mp;\ c_2 := 0 \text{ fi}; \\
\quad\quad \text{if } v_2^2 < 2b(\mathrm{MA}_2 - p_2) \text{ then } a_2 := A \text{ else } a_2 := 0 \text{ fi}\big\rangle \\
\quad\quad \langle \text{if } r < \varepsilon \text{ then } s := 2 \text{ else } s := 1 \text{ fi}\rangle \\
\quad\quad [\dot{p}_1 = s \cdot v_1, \dot{v}_1 = s \cdot a, \dot{c}_1 = s, v_1^2 \leq 2b(\mathrm{MA}_1 - p_1), c_1 \leq n \\
\quad\quad \dot{p}_2 = (2-s)v_2, \dot{v}_2 = (2-s)a, \dot{c}_2 = 2-s, v_2^2 \leq 2b(\mathrm{MA}_2 - p_2), \\
\quad\quad c_2 \leq n, \dot{r} = 2s - 2, |r| \leq \varepsilon]\Big)\Big)\Big) \\
\cap[?true]\Big)\ \langle ?x_1 = x_2 \wedge v_1 = v_2 \wedge m_1 = m_2\rangle\Big)^{[*]}
\end{array}
$$

Figure 7.7: Robust refinement game for the ETCS model

The proof has 204465 steps on 2083 branches. With a single interactions (supplying the loop invariant) the proof takes roughly 48h to complete using Redlog for local quantifier elimination and Z3 to deal with the resulting purely universal proof obligations. Instantiating some of the parameters with for example $n = 7$, $mp = 10$, $A = 1$, and $b = 3$ the proof finishes within 157 minutes (235508 steps, 2083 branches).

Let $\varepsilon = com$ for $com \geq 0$. For the initial regions

$$x_1 = 0 \land v_1 = 0 \land m_1 = 0 \land c_1 = -com$$

and

$$x_2 = 0 \land v_2 = 0 \land m_2 = 0 \land c_2 = 0 \ ,$$

we can thus show that Verifier has a winning strategy in the game depicted in Figure 7.7. Therefore, we know that $I_{com} \wr \xrightarrow{com,0} S$ by Theorem 9 (see page 149). Hence with $re_{com,0}(\Box(p \leq \text{MA})) = \Box(p \leq \text{MA})$ we can conclude from

$$S \models \Box(p \leq \text{MA})$$

that

$$I_{com} \models \Box(p \leq \text{MA}) \ .$$

7.4 Conclusion

In this chapter we have utilized the ETCS as a case study for our approach advocated in the Chapters 4 and 5. We have modeled a simple train controller interacting with an RBC and proven that it is safe in the sense that it always stays within its movement authority. Subsequently, we have developed an implementation template that suffers from communication delays in the communication with the RBC. Using hybrid games and dDG$\mathcal{L}$ we were able to show that these implementations are robust refinements of our original model. That way, we reasoned that our safety results from this specification transfer to its implementations. Our model in this system used very simple continuous dynamics. As the rules from the DAL calculus are implemented in KeYmaera adding more realistic train dynamics would be feasible. An interesting extension to the case study would be considering non-constant communication delays. In that case we would assume that some more assumptions on the relation of the communication delay, the cycle time of the RBC, the length of the movement authority extension, and acceleration bounds would be needed.

Note that, the point of this case study is to show that the tools developed throughout this thesis can be applied to prove robust refinement of hybrid systems. Of course, our method is more beneficial in cases where there is a large number of properties that have been shown for the specification. In the example discussed in this chapter, it would obviously be much easier to prove safety for the implementations directly instead of concluding it from a large proof for the robust refinement relation. Still, we have demonstrated that showing similarity of hybrid systems can be done effectively using our methods. Observe that, in cases where n is small and mp is large the train never reaches the end of its movement authority again. This means for that case there is no δ such that some implementation with $com > 0$ is an 0-δ-refinement as the distance between the two trajectories would strictly monotonically increasing (it would follow some second order function on the interval $[n, \infty[$ for the specification and the interval $[n + com, \infty[$ for the implementation).

CHAPTER EIGHT

Conclusion

Coming back to where you started
is not the same as never leaving.
— Terry Pratchett

Contents

In this chapter we summarize the results of this thesis, give some concluding remarks w.r.t. straight forward extensions, and discuss direction for future work.

8.1 Summary

In this thesis we have studied several different families of similarity notions. Thereby, we have built a formal basis to relate different models and transfer properties that have been proven for one system to another. One well-studied property type is stability. We have shown that each robust refinement of a stable specification is region stable. Further, we presented a variant of metric temporal logic, our new logic $\mathcal{L}\natural$ (natural logic), to formulate more complex properties of hybrid systems. Subsequently, we have given a syntactic transformation that computes from a set of properties of a specification a set of properties that all robust refinements of this specification exhibit. In order to prove that two systems are similar we have studied hybrid games and presented a second new logic, differential dynamic game logic ($\mathsf{dDG}\mathcal{L}$), to reason about these. Based on automata and program representations of hybrid systems, we have given two approaches to show that two systems are in robust refinement relation. We presented a sequent proof calculus for $\mathsf{dDG}\mathcal{L}$ which we implemented in our theorem prover KeYmaera. As a showcase for $\mathsf{dDG}\mathcal{L}$ we considered a case study in which a robot plays a game against other agents in a factory automation scenario. A crucial part of proving properties about hybrid systems, hybrid games, or mathematical textbook algorithms is dealing with the resulting proof obligations in first-order logic over the reals. Therefore, we have taken a deeper look into different methods of proving real arithmetic statements. We presented a new approach for showing the validity of universal first-order formulas over the reals based on Gröbner Bases and semidefinite programming for the Real Nullstellensatz that produces checkable certificates. Subsequently, we provided experimental evidence that proof producing methods for this task make a lot of progress and even outperform some well-established tools on an interesting set of benchmarks. In order to demonstrate the applicability of our approach for showing robust refinement, we have considered a case study from the domain of train control and proven that a system struggling with communication delays is a robust refinement of a specification that assumes instantaneous message passing. That way, we were able to reason that as the specification is safe so are its imperfect implementations.

8.2 Concluding Remarks

In this thesis we formulated our notions of similarity in terms of hybrid programs. Note that we could alternatively define these notions in terms of quantified hybrid programs [Pla10c] as well. Quantified hybrid programs can be used to express dynamically reconfigurable hybrid systems. In order to show that two such systems are in robust refinement relation, we can use the same constructions as in Chapter 5 on a variant of $\mathsf{dDG}\mathcal{L}$ that is based on quantified differential dynamic logic ($\mathsf{Qd}\mathcal{L}$) [Pla10c]. This extension of $\mathsf{dDG}\mathcal{L}$ towards dynamically reconfigurable system description can be done easily as the method to "gamify" dynamic logics presented in Chapter 4 on the example of $\mathsf{d}\mathcal{L}$ works on arbitrary dynamic logics and can thus be used easily to "gamify" $\mathsf{Qd}\mathcal{L}$. A pitfall that one should keep in mind is that the state space described by quantified hybrid systems is not finite dimensional. This does not pose an issue for the similarity notions itself. However, in order to compute the properties that are preserved under this extended notion of similarity we have to keep in mind that the equivalence of norms stated in Lemma 3 does not hold for infinite vector spaces. Therefore, we would need to choose a fixed norm like the maximum norm in order to be able to easily compute it component wise while still having nice properties like monotonicity. Other than that this extension is straightforward.

8.3 Future Work

In this section, we like to discuss some possible directions of future work. Certainly worthwhile to explore would be how we could extend our logic $\mathcal{L}\natural$ while still being able to transfer properties. Furthermore, we sketch the possibility to replace the constant bounds on the temporal and/or spatial distances by more general functions. As proving similarity of hybrid systems is computationally expensive, we discuss approaches for compositional reasoning in the sense of deducing from the similarity of parts of systems that the whole systems are similar. For our differential dynamic game logic the question arises whether we could extend it to reasoning about differential games in addition to deterministic continuous evolutions along differential equations. We sketch some ideas how this could be done and what possible proof rules could look like.

8.3.1 Exploiting Conjunctions

The transformation function to compute what properties are preserved presented in Chapter 3 transformes each subformula individually. However, it could be worthwhile to study what knowledge about the variable valuations is provided by surrounding formulas. For example if we have a formula like

$$\Box(x \in [2,4] \wedge x^2 - 3 \leq 0) \tag{8.1}$$

for all trajectories in a distance of at most 1 w.r.t. the Euclidean norm we can infer that they satisfy

$$\Box(x \in [1,5] \wedge (x-1)^2 - 3 \leq 0) \tag{8.2}$$

even though the function $f(x) = x^2 - 3$ is neither monotone nor Lipschitz continuous. This results from the fact that the function is monotone on the interval $[1,5]$. Thus, using the surrounding information gives us an edge over blindly checking the properties of the functions and disregarding this property as it does not match our conditions.

A similar construction could be done if we have knowledge about the Lipschitz continuity on a certain interval. Then, we could use the Lipschitz constant for the specific interval instead of a global one, which might overall give a tighter fit.

8.3.2 Dynamic Bounds

An interesting extension to our notions of similarity could be to move from constant bounds on the spatial and temporal distances to functions defining the bounds over time. Assume the spatial distance is bounded by a strictly decreasing monotone function that converges towards zero. In that case we could deduce from one system being asymptotically stable not only that its robust refinements are region stable but in addition use the fact that the variable valuations converge towards each other and, therefore, all robust refinements are asymptotically stable as well.

Definition 63 (Dynamic Similarity of Traces)**.** *For two traces σ_1 and σ_2, given a strictly monotonically decreasing function $\delta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, we say that σ_1 is* weakly $\delta(t)$-similar *to σ_2 (denoted by $\sigma_1 \xleftarrow{\infty}\!\!\wr\!\!\xrightarrow{\delta(t)} \sigma_2$) iff there is an retiming $\mathfrak{r}$ such that*

$$\forall (t, \tilde{t}) \in \mathfrak{r} : ||\sigma_1(t) - \sigma_2(\tilde{t})|| \leq \delta(t) \ .$$

In contrast to Definition 21 we have a strictly monotonically decreasing function defining the spatial bounds.

Conjecture 1. *Let σ_1 and σ_2 be hybrid traces such that $\sigma_1 \leftarrow\!\!\!\!\!\!\sim^{\infty}\!\!\!\!\!\!\rightsquigarrow^{\delta(t)} \sigma_2$. If σ_1 is asymptotically stable then σ_2 is asymptotically stable as well.*

8.3.3 Compositional Reasoning

To make proving robust refinement easier it would be nice to have compositional proof rules. However, it turns out that this task is rather difficult. For the choice operator we can use the following result to get a compositional proof rule.

Proposition 9. *If $A \mathrel{\wr\!\!\!\sim}^{\varepsilon_1,\delta_1} A'$ and $B \mathrel{\wr\!\!\!\sim}^{\varepsilon_2,\delta_2} B'$ then*

$$A \cup B \mathrel{\wr\!\!\!\sim}^{\max\{\varepsilon_1,\varepsilon_2\},\max\{\delta_1,\delta_2\}} A' \cup B' \ .$$

Proof. Assume $A \mathrel{\wr\!\!\!\sim}^{\varepsilon_1,\delta_1} A'$ and $B \mathrel{\wr\!\!\!\sim}^{\varepsilon_2,\delta_2} B'$. Consider an arbitrary hybrid trace $\sigma \in \tau(A \cup B)$ of $A \cup B$. From the semantics of $\cup$ we know that either $\sigma \in \tau(A)$ or $\sigma \in \tau(B)$. By definition, $\tau(A' \cup B') = \tau(A') \cup \tau(B')$. We make a case distinction over the origin of trace σ. If $\sigma \in \tau(A)$ then there is a trace $\sigma_{A'}$ of A' that is close because $A \mathrel{\wr\!\!\!\sim}^{\varepsilon_1,\delta_1} A'$ holds. Also, this traces can be chosen by $A' \cup B'$, i.e., $\sigma_{A'} \in \tau(A' \cup B')$. From Lemma 7 we know that our relation is monotone w.r.t. parameter values. Therefore, this trace does not yield a counterexample to our assumption that $A \cup B \mathrel{\wr\!\!\!\sim}^{\max\{\varepsilon_1,\varepsilon_2\},\max\{\delta_1,\delta_2\}} A' \cup B'$. The second case, i.e., $\sigma \in \tau(B)$, is symmetric. Overall, we can conclude that

$$A \cup B \mathrel{\wr\!\!\!\sim}^{\max\{\varepsilon_1,\varepsilon_2\},\max\{\delta_1,\delta_2\}} A' \cup B' \ .$$

□

However, already for sequential composition it gets more difficult. One could conjecture that like for our weak transitivity it holds that $A \mathrel{\wr\!\!\!\sim}^{\varepsilon_1,\delta_1} A'$ and $B \mathrel{\wr\!\!\!\sim}^{\varepsilon_2,\delta_2} B'$ imply $A; B \mathrel{\wr\!\!\!\sim}^{\varepsilon_1+\varepsilon_2,\delta_1+\delta_2} A'; B'$. However, this conjecture is false as we can see from the following example.

Example 27. *Let A, A', B, B' be defined as follows:*

$$A \mathrel{\hat{=}} x := 0; \dot{x} = 0$$
$$A' \mathrel{\hat{=}} x := 1; \dot{x} = 0$$
$$B \mathrel{\hat{=}} ?x = 0; \dot{x} = 1$$
$$B' \mathrel{\hat{=}} ?x = 0; \dot{x} = 1$$

We can easily see that $A; B$ has runs where x can reach any positive value. Further, $A \wr^{\varepsilon_1,\delta_1}\!\!\rightarrow A'$ for any $\varepsilon_1 \geq 0$ and any $\delta_1 \geq ||1||$. The systems B and B' are even identical. Still, the composition $A'; B'$ does not have any run and is, thus, not similar to $A; B$. This disproves our conjecture.

The issue is that we do not gain sufficient knowledge about B' with regard to its overall behavior as its initial values are existentially quantified in our notion of refinement. Hence we need stronger premises. Let $v_C := x$ be a vectorial assignment that updates all variables of a system C.

Proposition 10. *If $A \wr^{\varepsilon_1,\delta_1}\!\!\rightarrow A'$ and $v_B := a; B \wr^{\varepsilon_2,\delta_2}\!\!\rightarrow v_{B'} := b; B'$ holds for all points $(a, b) \in \mathbb{R}^n \times \mathbb{R}^n$ with $||a - b|| \leq \delta_1$ then*

$$A; B \wr^{\varepsilon_1+\varepsilon_2,\max\{\delta_1,\delta_2\}}\!\!\rightarrow A'; B' \ .$$

Proof. Assume that $A \wr^{\varepsilon_1,\delta_1}\!\!\rightarrow A'$ and $v_B := a; B \wr^{\varepsilon_2,\delta_2}\!\!\rightarrow v_{B'} := b; B'$ holds for all points $(a, b) \in \mathbb{R}^n \times \mathbb{R}^n$ with $||a - b|| \leq \delta_1$. Further, assume that there is a trace $\sigma \in \tau(A; B)$ as otherwise there is nothing to show. This means that there is a trace $\sigma_A \in \tau(A)$ of A and a trace $\sigma_B \in \tau(B)$ of B such that $last(\sigma_A) = first(\sigma_B)$ and $\sigma_A \circ \sigma_B = \sigma$. Let $\sigma_{A'} \in \tau(A')$ be such that $\sigma_A \leftarrow^{\varepsilon_1}\!\!\wr^{\delta_1}\!\!\rightarrow \sigma_{A'}$. This trace exists as we assume $A \wr^{\varepsilon_1,\delta_1}\!\!\rightarrow A'$ to hold. From this, we know that

$$||last(\sigma_A) - last(\sigma_{A'})|| \leq \delta_1 \ . \tag{8.3}$$

Further, we can find a trace $\sigma_{B'}$ with $first(\sigma_{B'}) = last(\sigma_{A'})$ such that it is similar to σ_B, i.e., $\sigma_B \leftarrow^{\varepsilon_2}\!\!\wr^{\delta_2}\!\!\rightarrow \sigma_{B'}$. This is because we assume

$$v_B := a; B \wr^{\varepsilon_2,\delta_2}\!\!\rightarrow v_{B'} := b; B'$$

holds for all points $(a, b) \in \mathbb{R}^n \times \mathbb{R}^n$ with $||a - b|| \leq \delta_1$. Hence we can combine these two traces $\sigma_{A'} \circ \sigma_{B'} = \sigma'$. The resulting trace is close to the original one, i.e., $\sigma \leftarrow^{\varepsilon_1+\varepsilon_2}\!\!\wr^{\max\{\delta_1,\delta_2\}}\!\!\rightarrow \sigma'$. This is, because on the first

part the distance is at most δ_1 and on the second part it is at most δ_2. Thus, overall it is at most $\max\{\delta_1, \delta_2\}$. The temporal deviations, however, add up. This results from the fact that it might be necessary to slow down one system on both parts of the trajectory. As σ was arbitrary, we can conclude that $A; B \mathrel{\wr\!\!\!\rightarrow}^{\varepsilon_1+\varepsilon_2, \max\{\delta_1,\delta_2\}} A'; B'$. □

Observe that the proposition has an uncountably many premisses. Still it would be possible to adapt our robust refinement game construction to this notion. This is, instead of letting Verifier choose the initial variable valuation for the variables in B' we give this choice to Falsifier with the restriction that the choice has to satisfy $||v_B - v_{B'}|| \leq \delta$.

For parallel composition we are out of luck for our notion of robust refinement. Let us consider the case of a parallel composition operator like defined on hybrid automata with synchronization on time passage and interleaving on discrete actions.

Example 28. *We extend Example 10 (see page 76). Let A, A', B, B' be defined as follows:*

$$A \mathrel{\hat{=}} x := 1; \dot{x} = x$$
$$A' \mathrel{\hat{=}} x := 2; \dot{x} = x$$
$$B \mathrel{\hat{=}} y := 2; \dot{y} = y$$
$$B' \mathrel{\hat{=}} y := 1; \dot{y} = y$$

As we can easily see it holds that $A \mathrel{\wr\!\!\!\rightarrow}^{\varepsilon,\delta} A'$ for $\varepsilon \geq \ln(1)$ and $\delta \geq ||1||$. The same holds for B and B', i.e. $B \mathrel{\wr\!\!\!\rightarrow}^{\varepsilon,\delta} B'$. However, there is no common retiming such that for all traces of $A \parallel B$ there is a trace of $A' \parallel B'$ that is close. This is, because for the similarity of A and A' we have to speed up the evolution of A. Whereas for B and B' we have to slow down the evolution of B. When running the systems in parallel, however, we cannot do both. Thus, it does not hold that $A \parallel B \mathrel{\wr\!\!\!\rightarrow}^{\varepsilon,\delta} A' \parallel B'$.

However, for the case where there are no temporal deviations, i.e., that of 0-δ-refinements, there might be hope. Let the norm be the maximum norm.

Proposition 11. *For disjoint parallelism it holds that if $A \mathrel{\wr\!\!\!\rightarrow}^{0,\delta_1} A'$ and $B \mathrel{\wr\!\!\!\rightarrow}^{0,\delta_2} B'$ then $A \parallel B \mathrel{\wr\!\!\!\rightarrow}^{0,\max\{\delta_1,\delta_2\}} A' \parallel B'$.*

Proof. Assume $A \wr^{0,\delta_1}\!\!\rightarrow A'$ and $B \wr^{0,\delta_2}\!\!\rightarrow B'$. Let $\sigma \in \tau(A \parallel B)$ be a trace of $A \parallel B$. By projection, we can extract a trace of A, i.e., $\sigma_A \in \tau(A)$ and one of B, i.e., $\sigma_B \in \tau(B)$ such that $\sigma_A \parallel \sigma_B = \sigma$ for a suitable definition of $\parallel$ on traces. Let $\sigma_{A'} \in \tau(A')$ be a trace such that $\sigma_A \leftarrow\!\!\wr^{0}\!\!\wr^{\delta_1}\!\!\rightarrow \sigma_{A'}$ and $\sigma_{B'} \in \tau(B')$ be a trace such that $\sigma_B \leftarrow\!\!\wr^{0}\!\!\wr^{\delta_2}\!\!\rightarrow \sigma_{B'}$. These traces exist by assumption. Let $\sigma_{A'} \parallel \sigma_{B'} =: \sigma' \in \tau(A' \parallel B')$ be a trace of $A' \parallel B'$. From our assumptions, we know that for each point in time the distance w.r.t. the variables of systems A and A' is at most δ_1. The distance w.r.t. the variables of systems B and B' is at most δ_2. Hence as the norm is the maximum norm, the overall distance is, therefore, bounded by $\max\{\delta_1, \delta_2\}$. Therefore, $\sigma \leftarrow\!\!\wr^{0}\!\!\wr^{\max\{\delta_1,\delta_2\}}\!\!\rightarrow \sigma'$ and, as σ was an arbitrary trace of $A \parallel B$, $A \parallel B \wr^{0,\max\{\delta_1,\delta_2\}}\!\!\rightarrow A' \parallel B'$. □

This may give rise to a compositional (w.r.t. the operations choice, sequential composition, and parallel composition) proof search for showing 0-δ-refinements.

8.3.4 Differential Dynamic Game Logic with Disturbances and Control

In our definition of dDG$\mathcal{L}$ we restricted the interactions between the players to discrete moves. However, in a continuous world influences by an analog controller and the environment can change the dynamics of a system concurrently. Therefore, we propose continuous disturbances and control inputs as a concept to model such behaviors. This extension basically replaces differential equation systems by differential games [Isa65], where the two players pursue different reachability goals.

Syntax. For this extension we propose to add the differential games beside the existing modalities.

$$M ::= [\alpha] \mid \langle\alpha\rangle \mid [\dot{x} = f(x,u,d), \chi]$$
$$G ::= M \mid (G_1 \cap G_2) \mid (G_1 \cup G_2) \mid (G_1 G_2) \mid (G)^{[*]} \mid (G)^{\langle * \rangle}$$

We conjecture that it is necessary to assume that f is globally Lipschitz continuous in x, continuous in u and d, and bounded like in the work of Gao et al. [GLQ07].

Semantics. For a function $f : X \to Y$ let $Z \lhd f$ for $Z \subseteq X$ denote the domain restriction of f to X.

Definition 64 (Control Law)**.** *Let $\mathfrak{d}$ denote the dimension of the disturbance inputs and $\mathfrak{u}$ the dimension of the control inputs. A* control law *is a family of functions $c_R : \mathbb{R}_{\geq 0} \times (R \to \mathbb{R}^{\mathfrak{d}}) \times \mathbb{R}^n \to (\mathbb{R}_{\geq 0} \to \mathbb{R}^{\mathfrak{u}})$ for $R \subseteq \mathbb{R}_{\geq 0}$ that satisfy the following properties:*

- *For all $t \in \mathbb{R}_{\geq 0}$, $x \in \mathbb{R}^n$, $d \in R \to \mathbb{R}^{\mathfrak{d}}$ with $R \subseteq \mathbb{R}_{\geq 0}$ the function $c_R(t, d, x)$ is Lebesgues-measurable.*
- *For all $T \geq 0$, $x \in \mathbb{R}^n$, disturbances $d_1 \in R_1 \to \mathbb{R}^{\mathfrak{d}}$ with $R_1 \subseteq \mathbb{R}_{\geq 0}$, and disturbances $d_2 \in R_2 \to \mathbb{R}^{\mathfrak{d}}$ with $R_2 \subseteq \mathbb{R}_{\geq 0}$ if $d_1(t) = d_2(t)$ for almost every $t \in [0, T]$ then $c_{R_1}(s, d_1, x)(t') = c_{R_2}(s, d_2, x)(t')$ for all $s \geq T$ and almost all times $t' \in [0, T]$.*

The latter restriction on the possible control laws is used to enforce causality, i.e., the control law may not react on events that are occurring in the future. Sometimes, this property is called *non-antipativity* [GLQ07].

We are now able to amend our semantics.

12. $\nu \models [\dot{x} = f(x, u, d), \chi]\phi$ iff there is a control law c_R such that $\omega \models \phi$ for all states ω such that there is a $t \in \mathbb{R}_{\geq 0}$ and $d : [0, t] \to \mathbb{R}^{\mathfrak{d}}$ such that there is a solution y of the initial value problem

$$\dot{x}(s) = f(x(s), u(s), d(s)) \wedge x(0) = \nu(x)$$

with $u(s) = c_{[0,s]}(t, [0, s] \lhd d, x)(s)$, $(y(t'), u(t'), d(t')) \models \chi$ for all $0 \leq t' \leq t$, and $y(t) = \omega$.

Proof Calculus. If there are only inputs under control of Falsifier we could use differential algebraic constraints [Pla10b] in order to argue about these disturbances already in dDG$\mathcal{L}$.

$$[\exists d : \dot{x} = f(x, d), -min \leq d \leq max]$$

For inputs under control of Verifier we have to do a more complex construction. As we announce the number of loop iterations in advance the maximum evolution time is known a priori by Verifier. In the following we make this explicit using g for the global upper bound.

If a piecewise constant input function for Verifier suffices then we could use the following encoding:

$$[t := 0; g := *; ?g \geq 0] \langle s := *;\ ?s > 0 \rangle \\ \big(\langle ?c < s \cup (u := *; c := 0)\rangle \\ \big[\dot{x} = f(x,u), \dot{c} = 1, \dot{t} = 1 \& c \leq s \wedge t \leq g\big]\big)^{[*]}$$

In this formula g denotes the maximum evolution time and this choice is binding for Falsifier which is governed by the clock t. The variables s denotes the sampling frequency of our input function and the clock c is used to ensure that Falsifier respects this choice.

We could extend this idea to inputs that are piecewise polynomial.

$$[t := 0; g := *; ?g \geq 0] \langle s := *;\ ?s > 0 \rangle \\ \left(\langle ?c < s \cup (a_0 := *; \cdots ; a_n := *; c := 0)\rangle \right. \\ \left. \left[\exists u : \dot{x} = f(x,u), \dot{c} = 1, \dot{t} = 1 \& u = \sum_{i=0}^{n} a_i x^i \wedge c \leq s \wedge t \leq g\right] \right)^{[*]}$$

In addition to the previous encoding of piecewise constant function, Verifier can choose coefficients of some polynomial over the state variables. The polynomial thus determined then describes the inputs for the next sampling period. We only sketch this for univariate polynomials here but it can be easily extended to the multivariate case.

This could even be extended further to piecewise polynomial differentiable inputs.

$$[t := 0; g := *; ?g \geq 0] \langle s := *;\ ?s > 0 \rangle \\ \left(\langle ?c < s \cup (a_0 := *; \cdots ; a_n := *; u := *; c := 0)\rangle \right. \\ \left. \left[\dot{x} = f(x,u), \dot{c} = 1, \dot{t} = 1, \dot{u} = \sum_{i=0}^{n} a_i x^i \& c \leq s \wedge t \leq g\right] \right)^{[*]}$$

Overall, this extension of dDG$\mathcal{L}$ certainly needs some more work. It could be worthwhile to consider as it might enable us to reason about

specifications which feature some slackness w.r.t. the continuous evolution. This slackness could be exploited when showing robust refinement relations. The simplest and most common form of slackness are differential inclusions. By allowing continuous inputs these could be added to our specification language.

Bibliography

[ABD08] Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors. *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2008. (Referenced on page(s) 220, 221, 225)

[ACHH92] 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. (Referenced on page(s) 4, 10, 11, 19, 25, 124)

[AD91] Rajeev Alur and David L. Dill. The theory of timed automata. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer-Verlag Berlin Heidelberg, 1991. (Referenced on page(s) 3, 4)

[AH92] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 74–106, London, UK, 1992. Springer-Verlag. (Referenced on page(s) 61)

[ALPK12] Nikos Aréchiga, Sarah M. Loos, André Platzer, and Bruce H. Krogh. Using theorem provers to guarantee closed-loop system properties. In Dawn Tilbury, editor, *American Control Conference, Montréal, Canada, June 27-29*, 2012. (Referenced on page(s) 11, 180)

[ÁMSH01] Erika Ábrahám-Mumm, Martin Steffen, and Ulrich Hannemann. Verification of hybrid systems: Formalization and proof rules in PVS. In *ICECCS*, pages 48–57. IEEE Computer, 2001. (Referenced on page(s) 158)

[AP07] Behzad Akbarpour and Lawrence C. Paulson. Extending a resolution prover for inequalities on elementary functions. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag Berlin Heidelberg, 2007. (Referenced on page(s) 183)

[BBC09] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier. O-minimal hybrid reachability games. *Logical Methods in Computer Science*, 6(1), 2009. (Referenced on page(s) 82, 116)

[BCR98] Jacek Bochnak, Michel Coste, and Marie-Francoise Roy. *Real Algebraic Geometry*, volume 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer-Verlag Berlin Heidelberg, 1998. (Referenced on page(s) 172)

[BHS07] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*, volume 4334 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2007. (Referenced on page(s) 156, 157)

[BLL+95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal - a tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems*, volume 1066 of *Lecture Notes in Computer*

Science, pages 232–243. Springer-Verlag Berlin Heidelberg, 1995. (Referenced on page(s) 4)

[Bor99] Brian Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11(1-4):613–623, 1999. (Referenced on page(s) 161, 174, 179)

[Bro07] Luitzen Egbertus Jan Brouwer. *Over de grondslagen der wiskunde*. Maas & van Suchtelen, 1907. (Referenced on page(s) 5)

[Bro03] Christopher W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM Bull.*, 37(4):97–108, 2003. (Referenced on page(s) 161, 170)

[Buc65] Bruno Buchberger. *An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal*. PhD thesis, University of Innsbruck, 1965. (Referenced on page(s) 161, 163, 167)

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge Univ. Press, 2004. (Referenced on page(s) 173, 176, 184)

[BZSH12] Richard Banach, Huibiao Zhu, Wen Su, and Runlei Huang. Continuous KAOS, ASM, and formal control system design across the continuous/discrete modeling interface: a simple train stopping application. *Formal Aspects of Computing*, pages 1–48, 2012. (Referenced on page(s) 78, 188)

[CH91] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991. (Referenced on page(s) 161, 170, 171, 182, 184)

[Col75] George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Barkhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer-Verlag Berlin Heidelberg, 1975. (Referenced on page(s) 5, 161, 170)

[Con90] J.B. Conway. *A Course in Functional Analysis*. Graduate Texts in Mathematics. Springer-Verlag Berlin Heidelberg, 1990. (Referenced on page(s) 34)

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM Press, 1971. (Referenced on page(s) 25)

[CRT09] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Requirements validation for hybrid systems. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2009. (Referenced on page(s) 188)

[dAFS09] Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Trans. Software Eng.*, 35(2):258–273, 2009. (Referenced on page(s) 77)

[Dav09] Jennifer M. Davoren. Epsilon-tubes and generalized skorokhod metrics for hybrid paths spaces. In Rupak Majumdar and Paulo Tabuada, editors, *Hybrid Systems: Computation and Control, 12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009, Proceedings*, volume 5469 of *Lecture Notes in Computer Science*, pages 135–149. Springer-Verlag Berlin Heidelberg, 2009. (Referenced on page(s) 77, 80)

[DDD+12] Werner Damm, Henning Dierks, Stefan Disch, Willem Hagemann, Florian Pigorsch, Christoph Scholl, Uwe Waldmann, and Boris Wirtz. Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. *Science of Computer Programming*, 77(10-11):1122–1150, 2012. (Referenced on page(s) 132, 158)

[DH88] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1/2):29–35, 1988. (Referenced on page(s) 5, 170)

[DM10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010, Proceedings*, volume 6246 of *Lecture Notes in Computer Science*, pages 92–106. Springer-Verlag Berlin Heidelberg, 2010. (Referenced on page(s) 78, 80)

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer-Verlag Berlin Heidelberg, 2008. (Referenced on page(s) 161, 171, 180)

[DMO+07] Werner Damm, Alfred Mikschl, Jens Oehlerking, Ernst-Rüdiger Olderog, Jun Pang, André Platzer, Marc Segelken, and Boris Wirtz. Automating verification of cooperation, control, and design in traffic applications. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems*, volume 4700 of *LNCS*, pages 115–169. Springer, 2007. (Referenced on page(s) 189)

[DS97] Andreas Dolzmann and Thomas Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bull.*, 31:2–9, 1997. (Referenced on page(s) 161, 171)

[DSW98] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. A new approach for automatic theorem proving in real geometry. *Journal of Automated Reasoning*, 21(3):357–380, 1998. (Referenced on page(s) 180)

[Ehr61] Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961. (Referenced on page(s) 152)

[EKM98] Jacob Elgaard, Nils Klarlund, and Anders Møller. Mona 1.x: New techniques for ws1s and ws2s. 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 *Lecture Notes in Computer Science*, pages 516–520. Springer-Verlag Berlin Heidelberg, 1998. (Referenced on page(s) 188)

[ERT02] ERTMS User Group, UNISIG. ERTMS/ETCS System requirements specification. http://www.era.europa.eu, 2002. Version 2.2.2. (Referenced on page(s) 187, 189)

[FGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011, Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer-Verlag Berlin Heidelberg, 2011. (Referenced on page(s) 132, 158)

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007. (Referenced on page(s) 132)

[FIJSS10] Johannes Faber, Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. Automatic verification of parametric specifications with complex topologies. In D. Méry and S. Merz, editors, *Integrated Formal Methods*, volume 6396 of *Lecture Notes in Computer Science*, pages 152–167. Springer-Verlag Berlin Heidelberg, 2010. (Referenced on page(s) 188)

[Fit96] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, 2nd edition, 1996. (Referenced on page(s) 21, 22, 24)

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. (Referenced on page(s) 118, 121)

[FLOQ11] Johannes Faber, Sven Linker, Ernst-Rüdiger Olderog, and Jan-David Quesel. Syspect - modelling, specifying, and verifying real-time systems with rich data. *International Journal of Software and Informatics*, 5(1-2):117–137, 2011. ISSN 1673-7288. (Referenced on page(s) 188)

[FM99] Melvin Fitting and Richard L Mendelsohn. *First-order modal logic*, volume 277. Springer, 1999. (Referenced on page(s) 6)

[For08] Otto Forster. *Analysis Band 2 Differential- und Integralrechnung im R^n, Gewöhnliche Differentialgleichungen*. Vieweg-Studium 31: Grundkurs Mathematik. Vieweg, Braunschweig, 8 edition, 2008. (Referenced on page(s) 29, 30)

[FP09] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009. (Referenced on page(s) 78)

[Fre05] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag Berlin Heidelberg, 2005. (Referenced on page(s) 158)

[Fre08] Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *Software Tools for Technology Transfer (STTT)*, 10(3):263–279, 2008. (Referenced on page(s) 132)

[FT11] Uli Fahrenberg and Stavros Tripakis, editors. *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings*, volume 6919 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2011. (Referenced on page(s) 226, 228)

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift*, 39, 1935. (Referenced on page(s) 99)

[GJP08] Antoine Girard, A. Agung Julius, and George J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2):163–179, 2008. (Referenced on page(s) 76, 77)

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman, Boston, MA, USA, 1994. (Referenced on page(s) 179)

[GLQ07] Yan Gao, John Lygeros, and Marc Quincampoix. On the Reachability Problem for Uncertain Hybrid Systems. *IEEE Transactions on Automatic Control*, 52(9), September 2007. (Referenced on page(s) 117, 206, 207)

[GMS12] Bernhard Gramlich, Dale Miller, and Uli Sattler, editors. *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2012. (Referenced on page(s) 220, 226)

[GNRR93] Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 1993. (Referenced on page(s) 211, 221)

[Göd33] Kurt Gödel. Eine Interpretation des intuitionistischen Aussagenkalküls. *Ergebnisse eines mathematischen Kolloquiums*, 4:34–38, 1933. (Referenced on page(s) 5)

[Har07] John Harrison. Verifying nonlinear real formulas via sums of squares. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732 of *Lecture Notes in Computer Science*, pages 102–118. Springer-Verlag Berlin Heidelberg, 2007. (Referenced on page(s) 161, 168, 172, 174, 175, 179, 182, 183, 184)

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996,*

Proceedings, pages 278–292, Los Alamitos, 1996. IEEE Computer Society. (Referenced on page(s) 4, 10, 11, 19)

[HHM99] Thomas A. Henzinger, Benjamin Horowitz, and Rupak Majumdar. Rectangular hybrid games. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 1999. (Referenced on page(s) 82, 116)

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer (STTT)*, 1(1-2):110–122, 1997. (Referenced on page(s) 4, 11, 158)

[Hil93] David Hilbert. Ueber die vollen Invariantensysteme. *Mathematische Annalen*, 42(3):313–373, 1893. (Referenced on page(s) 164)

[HKPV95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 373–382. ACM, 1995. (Referenced on page(s) 4)

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic logic*. MIT Press, 2000. (Referenced on page(s) 6, 28)

[HMP05] Thomas A. Henzinger, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying similarities between timed systems. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, volume 3829 of *Lecture Notes in Computer Science*, pages 226–241. Springer-Verlag Berlin Heidelberg, 2005. (Referenced on page(s) 77)

[Hoa78] Charles Anthony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978. (Referenced on page(s) 188)

[Isa65] Rufus Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. SIAM series in applied mathematics. Wiley, 1965. (Referenced on page(s) 206)

[JdM12] Dejan Jovanovic and Leonardo Mendonça de Moura. Solving non-linear arithmetic. In Gramlich et al. [GMS12], pages 339–354. (Referenced on page(s) 161, 171)

[JN91] Charles R. Johnson and Peter Nylen. Monotonicity properties of norms. *Linear Algebra and its Applications*, 148(0):43 – 58, 1991. (Referenced on page(s) 32)

[Kha96] H.K. Khalil. *Nonlinear System*. Prentice Hall, 1996. (Referenced on page(s) 48, 49, 50)

[Kov08] Laura Kovács. Aligator: A mathematica package for invariant generation (system description). In Armando et al. [ABD08], pages 275–282. (Referenced on page(s) 180)

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. (Referenced on page(s) 6, 61)

[Lew18] Clarence Irving Lewis. *A Survey of Symbolic Logic*. Semicentennial publications of the University of California, 1868-1918. University of California Press, 1918. (Referenced on page(s) 5)

[LPN11] Sarah M. Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *FM*, volume 6664 of *Lecture Notes in Computer Science*, pages 42–56. Springer-Verlag Berlin Heidelberg, 2011. (Referenced on page(s) 180)

[MH05] Sean McLaughlin and John Harrison. A proof-producing decision procedure for real arithmetic. In Nieuwenhuis [Nie05], pages 295–314. (Referenced on page(s) 161, 171, 183, 184)

[Mil99] R. Milner. *Communicating and Mobile Systems: The Pi Calculus*. Cambridge University Press, 1999. (Referenced on page(s) 45)

[MLP12] Stefan Mitsch, Sarah M. Loos, and André Platzer. Towards formal verification of freeway traffic control. In Chenyang Lu, editor, *ACM/IEEE Third International Conference on Cyber-Physical Systems, Beijing, China on April 17 - 19*, 2012. (Referenced on page(s) 180)

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In Ernst W. Mayr and Claude Puech, editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995. (Referenced on page(s) 82)

[MRO02] Thomas Moor, Jörg Raisch, and Siu O'Young. Discrete supervisory control of hybrid systems based on l-complete approximations. *Discrete Event Dynamic Systems*, 12(1):83–107, 2002. (Referenced on page(s) 78)

[MS98] Zohar Manna and Henny Sipma. Deductive verification of hybrid systems using STeP. 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 *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 1998. (Referenced on page(s) 158)

[Nie05] Robert Nieuwenhuis, editor. *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2005. (Referenced on page(s) 220, 228)

[Nip08] Tobias Nipkow. Linear quantifier elimination. In Armando et al. [ABD08], pages 18–33. (Referenced on page(s) 183, 184)

[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. (Referenced on page(s) 4, 10, 11, 19, 124)

[NRS01] Andreas Nonnengart, Georg Rock, and Werner Stephan. Using hybrid automata to express realtime properties in VSE-II. In Ingrid Russell and John F. Kolen, editors, *FLAIRS*. AAAI Press, 2001. (Referenced on page(s) 158)

[Orl28] Ivan Efimovich Orlov. The Calculus of Compatibility of Propositions (in Russian). *Mathematics of the USSR. Sbornik (Matematicheskii Sbornik)*, 35:263–286, 1928. (Referenced on page(s) 5)

[OW08] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS '08, pages 1–13, Berlin, Heidelberg, 2008. Springer-Verlag. (Referenced on page(s) 6)

[Par85] Rohit Parikh. The logic of games and its applications. In *Annals of Discrete Mathematics*, pages 111–140. Elsevier, 1985. (Referenced on page(s) 118, 120)

[Par03] Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, 96(2):293–320, 2003. (Referenced on page(s) 161, 168, 172, 173, 174, 178, 182, 183)

[PBGM62] Lev S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes*. Wiley Interscience, New York, 1962. (Referenced on page(s) 131)

[PC08] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, volume 5123 of *Lecture Notes in Computer Science*, pages 176–189. Springer-Verlag Berlin Heidelberg, Jul 2008. (Referenced on page(s) 184)

[PGHD04] Jan Peleska, Daniel Große, Anne Elisabeth Haxthausen, and Rolf Drechsler. Automated verification for train con-

trol systems. In *FORMS/FORMAT*, 2004. (Referenced on page(s) 188)

[Pla07a] André Platzer. Combining deduction and algebraic constraints for hybrid system analysis. In Bernhard Beckert, editor, *4th International Verification Workshop VERIFY'07, at CADE-21, Bremen, Germany, July 15-16, 2007*, volume 259 of *CEUR Workshop Proceedings*, pages 164–178. CEUR-WS.org, 2007. (Referenced on page(s) 158, 170, 184)

[Pla07b] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In Nicola Olivetti, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, 16th International Conference, TABLEAUX 2007, Aix en Provence, France, July 3-6, 2007, Proceedings*, volume 4548 of *Lecture Notes in Computer Science*, pages 216–232. Springer-Verlag Berlin Heidelberg, Jul 2007. (Referenced on page(s) 157)

[Pla07c] André Platzer. A temporal dynamic logic for verifying hybrid system invariants. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science, 5th International Symposium, LFCS'07, New York, USA, June 4-7, 2007, Proceedings*, volume 4514 of *Lecture Notes in Computer Science*, pages 457–471. Springer-Verlag Berlin Heidelberg, Jun 2007. (Referenced on page(s) 17, 29, 121)

[Pla08] André Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, 2008. (Referenced on page(s) 7, 14, 15, 99, 103, 105, 157)

[Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *Journal of Logic and Computation*, 20(1):309–352, 2010. (Referenced on page(s) 14, 15, 18, 19, 121, 157, 160)

[Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer-Verlag Berlin Heidelberg, Berlin Heidelberg, Sep 2010. (Referenced on page(s) 6, 10, 11, 14, 15, 18, 19, 27, 29, 99, 100, 103, 104, 105, 107, 108, 136, 141, 157, 160, 180, 207)

[Pla10c] André Platzer. Quantified differential dynamic logic for distributed hybrid systems. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010, Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag Berlin Heidelberg, 2010. (Referenced on page(s) 121, 201)

[Pla11] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23, 23nd International Conference on Automated Deduction, Wrocław, Poland, July 31 – August 5, 2011, Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 431–445. Springer-Verlag Berlin Heidelberg, 2011. (Referenced on page(s) 82)

[Pla12a] André Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, 8(4):1–44, 2012. Special issue for selected papers from CSL'10. (Referenced on page(s) 157)

[Pla12b] André Platzer. Differential game logic for hybrid games. Technical Report CMU-CS-12-105, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, March 2012. (Referenced on page(s) 118, 120)

[Pla12c] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. (Referenced on page(s) 160)

[Pla13] André Platzer. A complete axiomatization for differential game logic for hybrid games. Technical Report CMU-CS-13-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January 2013. (Referenced on page(s) 120)

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*,

pages 46–57. IEEE Computer Society, 1977. (Referenced on page(s) 6)

[PP03] Marc Pauly and Rohit Parikh. Game logic - an overview. *Studia Logica*, 75(2):165–182, 2003. (Referenced on page(s) 118, 121)

[PQ08a] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Armando et al. [ABD08], pages 171–178. (Referenced on page(s) 8, 83, 113, 151, 156)

[PQ08b] André Platzer and Jan-David Quesel. Logical verification and systematic parametric analysis in train control. In Magnus Egerstedt and Bud Mishra, editors, *HSCC*, volume 4981 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2008. (Referenced on page(s) 8, 11, 180, 188, 189)

[PQ09a] André Platzer and Jan-David Quesel. European Train Control System: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brasil, December 9-12, 2009. Proceedings*, volume 5885 of *Lecture Notes in Computer Science*, pages 246–265. Springer-Verlag Berlin Heidelberg, 2009. (Referenced on page(s) 8, 11, 180, 188, 189)

[PQ09b] André Platzer and Jan-David Quesel. European train control system: A case study in formal verification. Report 54, SFB/TR 14 AVACS, September 2009. ISSN: 1860-9821, avacs.org. (Referenced on page(s) 8, 189, 190)

[PQR09a] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In Renate A. Schmidt, editor, *International Conference on Automated Deduction, CADE'09, Montreal, Canada, Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 485–501. Springer-Verlag Berlin Heidelberg, 2009. (Referenced on page(s) 8, 156, 161, 162, 169, 178)

[PQR09b] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. Reports of SFB/TR 14 AVACS 52, SFB/TR 14 AVACS, June 2009. ISSN: 1860-9821, http://www.avacs.org. (Referenced on page(s) 165)

[Pra76] Vaughan R. Pratt. Semantical considerations on floyd-hoare logic. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 109–121. IEEE Computer Society, 1976. (Referenced on page(s) 6, 118, 121)

[Pri55] Arthur Norman Prior. *Time and modality*. John Locke lectures. Greenwood Press, 1955. (Referenced on page(s) 5)

[Pri67] Arthur Norman Prior. *Past, Present and Future*. Oxford books. Clarendon Press, 1967. (Referenced on page(s) 6)

[PW06] Andreas Podelski and Silke Wagner. Model checking of hybrid systems: From reachability towards stability. In João P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings*, volume 3927 of *Lecture Notes in Computer Science*, pages 507–521. Springer-Verlag Berlin Heidelberg, 2006. (Referenced on page(s) 51)

[QFD11] Jan-David Quesel, Martin Fränzle, and Werner Damm. Crossing the bridge between similar games. In Tripakis and Fahrenberg [FT11], pages 160–176. (Referenced on page(s) 8, 43, 44, 45, 82, 124)

[QP12a] Jan-David Quesel and André Platzer. Playing hybrid games with KeYmaera. In Gramlich et al. [GMS12], pages 439–453. (Referenced on page(s) 8, 83)

[QP12b] Jan-David Quesel and André Platzer. Playing Hybrid Games with KeYmaera. Reports of SFB/TR 14 AVACS 84, SFB/TR 14 AVACS, April 2012. ISSN: 1860–9821, http://www.avacs.org. (Referenced on page(s) 83)

[QS06] Jan-David Quesel and Andreas Schäfer. Spatio-temporal model checking for mobile real-time systems. In Kamel

Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings*, volume 4281 of *Lecture Notes in Computer Science*, pages 347–361. Springer-Verlag Berlin Heidelberg, 2006. (Referenced on page(s) 188)

[Rat06] Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Log.*, 7(4):723–748, 2006. (Referenced on page(s) 183)

[Rav13] Anders P. Ravn. Personal communication, 2013. (Referenced on page(s) 3)

[Rey83] Osborne Reynolds. An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels. *Philosophical Transactions of the Royal Society of London*, 174:pp. 935–982, 1883. (Referenced on page(s) 3)

[RLP11] David W. Renshaw, Sarah M. Loos, and André Platzer. Distributed theorem proving for distributed hybrid systems. In Shengchao Qin and Zongyan Qiu, editors, *Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings*, volume 6991 of *Lecture Notes in Computer Science*, pages 356–371. Springer-Verlag Berlin Heidelberg, 2011. (Referenced on page(s) 159)

[RS07] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Journal in Embedded Computing Systems*, 6(1), 2007. (Referenced on page(s) 132)

[Rüm07] Philipp Rümmer. A sequent calculus for integer arithmetic with counterexample generation. In Bernhard Beckert, editor, *4th International Verification Workshop VERIFY'07, at CADE-21, Bremen, Germany, July 15-16, 2007*, volume 259 of *CEUR-WS.org*, 2007. (Referenced on page(s) 167, 168)

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986. (Referenced on page(s) 167)

[Sch06] Andreas Schäfer. *Specification and Verification of Mobile Real-Time Systems*. PhD thesis, University of Oldenburg, December 2006. (Referenced on page(s) 188)

[SHL11] Sriram Sankaranarayanan, Hadjar Homaei, and Clayton Lewis. Model-based dependability analysis of programmable drug infusion pumps. In Fahrenberg and Tripakis [FT11], pages 317–334. (Referenced on page(s) 11)

[Smi00] Graeme Smith. *The Object Z Specification Language*. Advances in Formal Methods Series. Kluwer Academic Publisher, 2000. (Referenced on page(s) 188)

[SRKC00] B. Izaias Silva, Keith Richeson, Bruce H. Krogh, and Alongkrit Chutinan. Modeling and verification of hybrid dynamical system using CheckMate. In *ADPM 2000: 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems, September 18-19, 2000, Dortmund, Germany*, 2000. (Referenced on page(s) 158)

[SS99] João P. Marques Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999. (Referenced on page(s) 171)

[SS05] Viorica Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In Nieuwenhuis [Nie05], pages 219–234. (Referenced on page(s) 188)

[Sta02] Thomas Stauner. Discrete-time refinement of hybrid automata. In Claire J. 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 *Lecture Notes in Computer Science*, pages 407–420. Springer-Verlag Berlin Heidelberg, 2002. (Referenced on page(s) 79)

[Ste73] Gilbert Stengle. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Math. Ann.*, 207(2):87–97, 1973. (Referenced on page(s) 161, 168, 172)

[Str06] Adam W. Strzebonski. Cylindrical algebraic decomposition using validated numerics. *Journal Symbolic Computation*, 41(9):1021–1038, 2006. (Referenced on page(s) 171)

[Tab09] Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer London, Limited, 2009. (Referenced on page(s) 75, 153)

[Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 2nd edition, 1951. (Referenced on page(s) 5, 160, 169)

[TFL10] Claus R. Thrane, Uli Fahrenberg, and Kim G. Larsen. Quantitative analysis of weighted transition systems. *J. Log. Algebr. Program.*, 79(7):689–703, 2010. (Referenced on page(s) 75)

[Tiw05] Ashish Tiwari. An algebraic approach for the unsatisfiability of nonlinear constraints. In C.-H. Luke Ong, editor, *CSL*, volume 3634 of *Lecture Notes in Computer Science*, pages 248–262. Springer-Verlag Berlin Heidelberg, 2005. (Referenced on page(s) 183)

[TLS00] Claire J. Tomlin, John Lygeros, and Shankar Sastry. A Game Theoretic Approach to Controller Design for Hybrid Systems. *Proceedings of IEEE*, 88:949–969, July 2000. (Referenced on page(s) 82, 117)

[VPVD11] Vladimeros Vladimerou, Pavithra Prabhakar, Mahesh Viswanathan, and Geir Dullerud. Specifications for decidable hybrid games. *Theoretical Computer Science*, 412(48):6770 – 6785, 2011. (Referenced on page(s) 82, 116, 117)

[WAHKM03] Jr. Warren A. Hunt, Robert Bellarmine Krug, and J. Strother Moore. Linear and nonlinear arithmetic in ACL2. In *Proceedings, Correct Hardware Design and Verification Methods, 12th IFIP Conference*, volume 2860 of *Lecture Notes in Computer Science*, pages 319–333. Springer-Verlag Berlin Heidelberg, 2003. (Referenced on page(s) 183)

[Web40] Moritz Weber. Das Ähnlichkeitsprinzip der Physik und seine Bedeutung für das Modellversuchswesen. *Forschung auf dem*

Gebiet des Ingenieurwesens A, 11:49–58, 1940. (Referenced on page(s) 3)

[Wei97] Volker Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput.*, 8(2):85–101, 1997. (Referenced on page(s) 161, 170, 171)

[Wol03] Stephen Wolfram. *The Mathematica book (5. ed.)*. Wolfram-Media, 2003. (Referenced on page(s) 156, 157, 161, 167, 171)

[ZHR91] Zhou Chaochen, Charles Anthony Richard Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991. (Referenced on page(s) 188)

Index

Technical Reports

Fakultät II, Department für Informatik, Universität Oldenburg,
Postfach 2503, 26111 Oldenburg, Germany

1/87 A. Viereck: „Klassifikationen, Konzepte und Modelle für den Mensch-Rechner-Dialog“ (Dissertation)

2/87 A. Schwill: „Forbidden subgraphs and reduction systems: A comparison“

3/87 J. Kämper: „Non-uniform proof systems: A new framework to describe non-uniform and probabilistic complexity classes“

1/88 K. Ambos-Spies, H. Fleischhack, H. Huwig: „Diagonalizing over deterministic polynomial time“

2/88 A. Schwill: „Shortest edge-disjoint paths in geodetically connected graphs“

3/88 V. Claus, U. Lichtblau (Hrsg.): „1. Tagung zur Küsten-Informatik“

1/89 U. van der Valk: „Einige Entscheidbarkeits- und Unentscheidbarkeitsresultate für Klasse von S/T-Netzen unter Maximum Firing Strategie und unter Prioritätenstrategien“

2/89 J. Kämper: „Strukturelle Untersuchungen im Umfeld der Komplexitätsklassen P und NP unter besonderer Berücksichtigung nichtuniformer, probabilistischer und disjunktiv selbstreduzierender Algorithmen“ (Dissertation)

3/89 J. Kämper: „Nondeterministic oracle Turing machines with maximal computation paths“

1/90 A. Schwill: „Shortest edge-disjoint paths in graphs“ (Dissertation)

2/90 K.R. Apt, E.-R. Olderog: „Using transformations to verify parallel programs“

3/90 U. Lichtblau: „Flußgraphgrammatiken“ (Dissertation)

4/90 K.R. Apt, E.-R. Olderog: „Introduction to program verification“

5/90 H. Jasper: „Datenbankunterstützung für Prolog-Programmierumgebungen“ (Dissertation)

1/91 F. Korf: „Net-based efficient simulation of AADL specifications“

2/91 S.V. Krishnan, C. Pandu Rangan, A. Schwill, S. Seshadri: „Two disjoint paths in chordal graphs“

3/91 H. Eirund: „Modellierung und Manipulation multimedialer Dokumente“ (Dissertation)

4/91 G. Schreiber: „Ein funktionaler Äquivalenzbegriff für den hierarchischen Entwurf von Netzen“

1/92 A. Viereck (Hrsg.): „Ergebnisse der 11. Arbeitstagung, Mensch-Maschine Kommunikation“

2/92 P. Gorny, U. Daldrup, H. Schwab: „Zwischenbilanz: Menschengerechte Gestaltung von Software“

3/92 E.-R. Olderog, St. Rössig, J. Sander, M. Schenke: „ProCoS at Oldenburg: The Interface between Specification Language and occam-like Programming Language“

4/92 F. Korf: „Synthesis of VHDL Test Environments form Temporal Logic Specifications“

5/92 W. Kowalk: „Konstruktorentechnik: Neue Methoden zur Mengenrechnung, Logikrechnung und Intervallrechnung“

1/93 Ch. Dietz, G. Schreiber: „Eine Termdarstellung für S/T-Netze“

2/93 J. Sauer: „Wissensbasiertes Lösen von Ablaufplanungsproblemen durch explizite Heuristiken“

3/93 M. Sonnenschein, U. Lichtblau (Hrsg.): „6. Kolloquium der Arbeitsgruppe Informatik-Systeme“

4/93 H. Fleischhack, U. Lichtblau, M. Sonnenschein, R. Wieting: „Generische Definition hierarchischer zeitbeschrifteter höherer Petrinetze“

5/93 F. Köster, L. Twele, R. Wieting, W. Ziegler: „Fallbeispiele zur Modellierung mit THORNetzen“

1/94 R. Götze: „Dialogmodellierung für multimediale Benutzerschnittstellen“

2/94 B. Müller: „PPO – Eine objektorientierte Prolog-Erweiterung zur Entwicklung wissensbasierter Anwendungssysteme“

3/94 W. Damm, A. Mikschl: „Projekt Entwurf und Implementierung eines Multithreaded RISC-Prozessors“

4/94 S. Rössig: „A Transformational Approach to the Design of Communicating Systems“ (Dissertation)

5/94 G. Schreiber: „Funktionale Äquivalenz von Petri-Netzen“ (Dissertation)

1/95 A. Gronewold, H. Fleischhack: „Language Preserving Reductions of Safe Petri-Nets“

2/95 H. Reineke: „Struktur und Verhalten von verteilten endlichen Automaten“ (Dissertation)

3/95 H. Behrends: „Beschreibung ereignisgesteuerter Aktivitäten in datenbankgestützten Informationssystemen“ (Dissertation)

4/95 U. M. Levens: „Computerunterstütztes Modellieren von Musikstücken mit Petri-Netzen: Das Mailänder Konzept“

1/96 M. Burke: „FDDI und ATM in multimedialen Anwendungsumgebungen“ (Dissertation)

2/96 I. Pitschke: „Interaktive Rekonstruktion geometrischer Modelle aus digitalen Bildern“ (Dissertation)

1/97 L. Bölke: „Ein akustischer Interaktionsraum für blinde Rechnerbenutzer“ (Dissertation)

2/97 S. Schöf: „Verteilte Simulation höherer Petrinetze“ (Dissertation)

1/98 S. Kleuker: „Inkrementelle Entwicklung von verifizierten Spezifikationen für verteilte Systeme“ (Dissertation)

2/98 J. Bohn: „Mechanical Support and Validation of a Design Calculus for Communicating Systems by a Logic-Based Proof System“ (Dissertation)

3/98 L. Köhler: „Fuzzy Geometrie und Anwendungen in der medizinischen Bildverarbeitung“ (Dissertation)

4/98 J. Helbig: „Linking Visual Formalisms: A Compositional Proof System for Statecharts Based on Symbolic Timing Diagrams“ (Dissertation)

5/98 G. Stiege: „Edge Partitions in Undirected Graphs“

6/98 A. Gerns: „Entwicklung und Bewertung von Objektmigrationsstrategien für verteilte Umgebungen“

7/98 M. Stadler: „Abstrakte Rechnernetzmodelle als Grundlage einer umfassenden Automatisierung des Netzmanagements – Konzepte und Sprachen zu ihrer Umsetzung“ (Dissertation)

8/98 M.-S. Steiner: „Lastverteilung in heterogenen Systemen“

9/98 Clemens Otte: „Fuzzy-Prototyp-Klassifikatoren und deren Anwendung zur automatischen Merkmalsselektion“

1/99 Juliane Vorndamme: „Die Auswirkungen rechtlicher Verpflichtungen auf die Software-entwicklung“

2/99 E. Best/K.M. Richter: „Relational Semantics Revisited“

3/99 J. S. Lie: „Einsatz von Objektmigrationssystemen zur Leistungssteigerung in verteilten Systemen“

4/99 Zweijahresbericht des Fachbereichs Informatik

5/99 Ingo Stierand, Olaf Maibaum, Björn Briel, Günther Stiege: „Cassandra – Generierung, Analyse und Simulation von eingebetteten Multiprozessor-Echtzeitsystemen“

6/99 Gunnar Wittich: „Ein problemorientierter Ansatz zum Nachweis von Realzeiteigenschaften eingebetteter Systeme“

7/99 Annegret Habel, Jürgen Müller, Detlef Plump: „Double-Pushout Graph Transformation Revisited“

8/99 Ingo Stierand: „Eine Konfigurationssprache zur Erstellung von Ambrosia/MP-Systemen“

9/99 Igor V. Tarasyuk: „Equivalences for Concurrent and Distributed Systems“

10/99 Eike Best, Alexander Lavrov: „Generalised Composition Operations for High-Level Petri-Nets“

11/99 Alexander Lavrov: „Enhancing Mixed Nonlinear Optimization: A Hybrid Approach“

12/99 Alexander Lavrov: „Hybrid Techniques in Discrete-Event System Modelling and Control: some Examples“

13/99 Eike Best, Raymond Devillers, Maciej Koutny: „Recursion and Petri Nets“

14/99 Eike Best, Raymond Devillers, Maciej Koutny: „The Box Algebra = Petri Nets + Process Expressions“

15/99 Eike Best, Harro Wimmel: „Reducing k-safe Petri Nets to Pomset-equivalent 1-safe Petri Nets“

16/99 Udo Brockmeyer: „Verifikation von STATEMATE Designs“ (Dissertation)

1/00 Henning Dierks: „Specification and Verification of Polling Real-Time Systems“ (Dissertation)

2/00 Clemens Fischer: „Combination and Implementation of Processes and Data: from CSP-OZ to Java“ (Dissertation)

3/00 Cheryl Kleuker: „Constraint Diagrams“ (Dissertation)

4/00 Thomas Thielke: „Linear-algebraische Methoden zur Beschreibung, Verfeinerung und Analyse gefärbter Petrinetze“ (Dissertation)

1/01 Günther Stiege: „Higher Decomposition in Undirected Graphs“ (Bericht)

2/01 Ute Vogel: Zweijahresbericht

3/01 Josef Tapken: „Model-Checking of Duration Calculus Specifications“ (Dissertation)

4/01 Björn Briel: „Analyse eingebetteter Systeme mittels verteilter Simulation“ (Dissertation)

5/01 Günther Stiege: „Standard Decomposition and Periodicity of Digraphs“ (Bericht)

6/01 Ingo Stierand: „Ambrosia/MP – Ein Echtzeitbetriebssystem für eingebettete Mehrprozessorsysteme“ (Dissertation)

1/02 Giorgio Busatto, Annegret Habel: „Improving the Quality of Hypertexts Using Graph Transformation“ (Bericht)

2/02 Giorgio Busatto: „Modeling Hyperweb Dynamics through Hierarchical Graph Transformation“ (Bericht)

3/02 Giorgio Busatto: „An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation“ (Dissertation)

4/02 Laila Kabous: „An Object Oriented Design methodology for hard real Time Systems: The OOHARTS approach“ (Dissertation)

1/03 Ute Vogel: „Zweijahresbericht“

2/03 Olaf Maibaum: „Bestimmung symbolischer Laufzeiten in eingebetteten Echtzeitsystemen“ (Dissertation)

3/03 Günther Stiege, Ingo Stierand: „Connectedness-Based Hierarchical Decomposition of Undirected Graphs“ (Bericht)

4/03 Willi Hasselbring, Susanne Petersen: „Standards für die medizinische Kommunikation und Dokumentation“ (Bericht)

5/03 Andreas Möller: „Eine virtuelle Maschine für Graphprogramme“ (Bericht)

6/03 Tom Bienmüller: „Reducing Complexity for the Verification of Statemate Designs“ (Bericht)

7/03 Sandra Steinert: „Graph Programs for Graph Algorithms“ (Bericht)

8/03 Jochen Klose: „Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior“ (Dissertation)

1/04 Jens Oehlerking: „Transformation of Edmonds' Maximum Matching Algorithm into a Graph Program“ (Bericht)

2/04 Sergej Alekseev: „Dienste Intelligenter Netze Graphentheoretische Methoden in der Kontrollflussanalyse“ (Bericht)

3/04 Giorgio Busatto: „GraJ: A System for Executing Graph Programs in Java“ (Bericht)

1/05 Sergej Alekseev: „Ablaufanalyse objektorientierter Echtzeitanwendungen mit graphentheoretischen Methoden“ (Dissertation)

2/05 Ute Vogel: „Zweijahresbericht“

3/05 Igor Tarasyuk: „Discrete time stochastic Petri box calculus“ (Bericht)

1/06 Henning Dierks: „Time, Abstraction and Heuristics“ (Habilitation)

2/06 Li Sek Su: „Full-Output Siphons and Deadlock-Freeness for Free Choice Petri Nets“ (Bericht)

3/06 Timo Warns: „Solving Consensus Using Structural Failure Models“ (Bericht)

4/06 Sergej Alekseev: „Graphentheoretische Methoden in der Ablaufanalyse objektorientierter Anwendungen“ (Dissertation)

5/06 Li Sek Su: „Some Considerations on the Foundation of NP-Completeness Theory“ (Bericht)

6/06 Li Sek Su: „Semitraps and Deadlock-Freeness for Reduced Asymmetric Choice Nets“ (Bericht)

7/06 Li Sek Su: „Algorithms of computing the Deadlock Markings Sets for Petri Nets“ (Bericht)

8/06 Annegret Habel, Karl-Heinz Pennemann, Arend Rensink: „Weakest Preconditions for High-Level Programs (Long Version)“ (Bericht)

9/06 Jochen Hoenicke: „Combination of Processes, Data, and Time“ (Dissertation)

10/06 Steffen Becker, Marco Boscovic, Abhishek Dhama, Simon Giesecke, Jens Happe, Wilhelm Hasselbring, Heiko Koziolek, Henrik Lipskoch, Roland Meyer, Margarethe Muhle, Alexandra Paul, Jan Ploski, Matthias Rohr, Mani Swaminathan, Timo Warns, Daniel Winteler: „Trustworthy Software Systems: A Discussion of Basic Concepts and Terminology“ (Bericht)

11/06 Christian Zuckschwerdt: „Ein System zur Transformation von Konsistenz- in Anwendungsbedingungen“ (Bericht)

01/07 Andreas Schäfer: „Specification and Verification of Mobile Real-Time Systems“ (Dissertation)

02/07 Günther Stiege: „General Graphs“ (Bericht)

03/07 Wolfgang Kowalk: „Integralrechnung“ (Bericht)

04/07 Karl Azab, Karl-Heinz Pennemann: „Type Checking C++ Template Instantiation by Graph Programs“ (Bericht)

01/08 Roland Meyer: „On depth and breath in the Pi-Calculus“ (Bericht)

02/08 Ingo Brückner: „Slicing Integrated Formal Specifications for Verification“ (Dissertation)

03/08 Ute Vogel: „2-Jahres-Bericht 2004 - 2006“ (Bericht)

04/08 Günther Stiege: „Summierbare Familien“ (Bericht)

05/08 Igor V. Tarasyuk: „Investigating equivalence relations in dtsPBC“ (Bericht)

01/09 Elke Wilkeit: „2-Jahres-Bericht 2007 - 2008“ (Bericht)

02/09 Roland Meyer: „Structural Stationarity in the pi-Calculus“ (Dissertation)

03/09 InformatikerInnen des Moduls Soft Skills: „E-Book Soft Skills 2008„ (Bericht)

04/09 Eike Best: „Separability in Persistent Petri Nets“ (Bericht)

01/10 Igor Tarasyuk: „Equivalence relations for behaviour-preserving reduction and modular performance evaluation in dtsPBC“ (Bericht)

02/10 Roman Dubtsov: „Timed Transition Systems with Independence and Marked Scott Domains: an Adjunction“ (Bericht)

01/11 Elena S. Oshevskaya: „Matching Equivalences on Higher Dimensional Automata Models“ (Bericht)

02/11 Elke Wilkeit: „2-Jahres-Bericht, 01.10.2008 - 30.09.2010“ (Bericht)

03/11 Johannes Faber: „Verification Architectures for Complex Real-Time Systems“ (Dissertation)

04/11 Igor V. Tarasyuk: „Equivalences for modular performance analysis in dtsPBC“ (Bericht)

01/12 Irina Virbitskaite, Nataliya Gribovskaya, Eike Best: „Some Evidence on the Consistency of Categorical Semantics for Timed Interleaving Behaviours“ (Bericht)

1/12 Günther Stiege: „Playing with Knuth's words.dat“ (Bericht)

02/12 Günther Stiege: „Flowerfree Finding of Maximum Matchings“ (Bericht)

01/13 Wolfgang Kowalk: „RunSort - Ein effizienter Sortieralgorithmus“ (Bericht)

02/13 Günther Stiege: „Cliques and Graphs of Type WORDS“ (Bericht)