Systematic Development of Hybrid Systems

Thomas Markus Stauner

Institut für Informatik
der Technischen Universität München

Systematic Development of Hybrid Systems

Thomas Markus Stauner

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ. Prof. (komm.) Dr. Eike Jessen, em.
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Manfred Broy
2.	Univ.-Prof. Dr. Georg Färber

Die Dissertation wurde am 27.06.2001 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 27.10.2001 angenommen.

Abstract

Hybrid embedded systems are systems that are characterized by involving discrete as well as continuous dynamics, such as digital software interacting with an analog environment. In the design of such systems, different time models and techniques from different disciplines—mainly computer science and control theory—are usually employed. An isolated consideration of the discrete and continuous parts of a hybrid system already at the beginning of the development process together with imprecisely defined interaction between these parts can lead to inappropriate or incorrect designs. Therefore, this thesis introduces formal, integrated notations and development techniques for hybrid systems which support development processes that postpone this partitioning of a hybrid system to the later development phases.

The introduced notations for data-flow and control-flow in hybrid systems precisely define the interaction of discrete and continuous dynamics in a hybrid system. They are visual and modular, and can be used for environment modeling, requirements analysis and specification of hybrid systems. For these notations, the thesis defines refinement techniques as transformations of the hybrid models. The techniques introduced aim at moving from an abstract model based on a dense time scale to a model with more detailed timing constraints that operates in discrete time. They are defined such that they can be used together with methods from numerical mathematics and control theory.

Further motivation for the refinement techniques stems from a formalization and classification of the properties required of hybrid systems. The classification reveals that trace inclusion is a suitable refinement notion for most central properties of hybrid systems. Furthermore, the detailed study of the properties and some of their proof methods, as contained in the thesis, fosters a better understanding of hybrid systems by computer scientists and formalizes parallels between computer science and control theory.

For all introduced elements of a design methodology for hybrid systems the thesis identifies their position and potential within the development process of hybrid systems.

Kurzfassung

Hybride eingebettete Systeme sind durch die Mischung von diskreter und kontinuierlicher Dynamik gekennzeichnet, wie sie etwa bei digitaler Software in Wechselwirkung mit einer analogen Umgebung vorkommt. Bei dem Entwurf solcher Systeme werden unterschiedliche Zeitmodelle und Techniken aus unterschiedlichen Disziplinen, vor allem aus der Informatik und der Regelungstechnik, eingesetzt. Wenn die diskreten und kontinuierlichen Anteile eines hybriden Systems bereits zu Beginn des Entwicklungsprozesses isoliert betrachtet werden, und wenn zudem die Wechselwirkung dieser Teile nicht präzise definiert ist, so können ungeeignete oder fehlerhafte Entwürfe entstehen. Diese Arbeit führt daher formale, integrierte Notationen und Entwicklungstechniken für hybride Systeme ein, die Entwicklungsprozesse unterstützen in denen eine solche Partitionierung hybrider Systeme erst in späteren Entwicklungsphasen stattfindet.

Die eingeführten Notationen für den Daten- und Kontrollfluß in hybriden Systemen definieren die Wechselwirkung zwischen diskreter und kontinuierlicher Dynamik in solchen Systemen präzise. Sie sind grafisch und modular und können zur Umgebungsmodellierung, zur Anforderungsanalyse und zur Spezifikation hybrider Systeme eingesetzt werden. Für diese Notationen werden in der Arbeit Verfeinerungstechniken zur Transformation hybrider Modelle definiert. Die eingeführten Techniken zielen auf die Migration von einem abstrakten Modell mit zugrundeliegender dichter Zeitachse zu einem Modell mit detaillierten Zeitanforderungen, das zeitdiskret arbeitet. Die Techniken erlauben es, sie zusammen mit Methoden der numerischen Mathematik und der Regelungstechnik anzuwenden.

Eine weitere Motivation der Verfeinerungstechniken resultiert aus einer Formalisierung und Klassifikation der von hybriden Systemen geforderten Eigenschaften. Die Klassifikation zeigt, daß Verhaltensinklusion ein geeigneter Verfeinerungsbegriff für die meisten zentralen Eigenschaften hybrider Systeme ist. Darüberhinaus fördert die in dieser Arbeit enthaltene detailierte Analyse der Eigenschaften und einiger ihrer Beweisverfahren ein besserer Verständnis hybrider Systeme durch Informatiker. Sie formalisiert zudem Parallelen zwischen Informatik und Regelungstechnik.

Alle in dieser Arbeit eingeführten Elemente einer Entwicklungsmethodik hybrider Systeme werden in den Entwicklungsprozeß hybrider Systeme eingeordnet, und ihr Potential wird aufgezeigt.

Acknowledgments

First of all I like to thank Manfred Broy for offering me the opportunity to join his group and for providing the atmosphere and support which enabled this work. Further thanks go to Georg Färber for undertaking the effort of being my referee in the dissertation committee and for our valueable discussions.

I like to thank Olaf Müller for giving me the impulse to start into the field of hybrid systems. I am particularly grateful to Radu Grosu for our intensive collaboration, which not only laid foundations for this thesis, but also helped me to learn how scientific work can be organized.

For stimulating discussions providing different points of view on hybrid systems I like to thank Christoph Grimm, István Péter, Thomas Schlegl and the team members of the MOBASIS project. Discussions with my colleagues and their comments on my work helped me to shape this thesis. Here, I primarily like to thank Ingolf Krüger, Jan Philipps, Wolfgang Prenninger and Bernhard Schätz. I am particularly indebted to Alexander Pretschner for loads of discussions and to Robert Sandner for his precise comments. Both gave me detailed feedback on large parts of this thesis at a point where time was tight. Further thanks go to Michel Sintzoff for his helpful hints and references on the study of properties of hybrid systems. Besides that I like to thank Heather Gumenik for her help to improve the English in this thesis. Furthermore, I thank the remaining colleagues in our group for the excellent atmosphere and infrastructure as well as all my friends for their patience.

I am deeply indebted to my parents for enabling me to go this way and helping me at so many points in my life. Finally, I thank Alexandra Gürtler who believed in me when I did not and who reminded me that there are more important things in life when I was about to forget. You helped me to put my work aside when I had spent more than enough time on it!

Contents

1 Introduction **1**

1.1 What are "Hybrid Systems"? 1

1.2 Subject of the Thesis 5

1.3 Structure of the Thesis 6

1.4 The Running Example 7

2 Methodology **9**

2.1 Preliminaries: Development of Control System 9

2.1.1 Development Steps 9

2.1.2 Important Notations 13

2.2 Hybrid Systems' Development 15

2.2.1 A Conventional Development Process 16

2.2.2 An Integrated Development Process 19

2.2.3 Supporting Techniques Developed in This Thesis 23

2.2.4 Other Techniques Supporting Integrated Development 25

2.3 Discussion and Further Work 27

2.3.1 Related Work 28

2.3.2 Further Work 29

3 HyCharts **31**

3.1 An Example 33

3.2 The Hybrid Computation Model 36

3.2.1 General Idea 36

3.2.2 The Discrete Part 39

3.2.3 The Analog Part . . . 40
3.2.4 Semantics of the Hybrid Computation Model . . . 42
3.2.5 A Note on Semantics . . . 44
3.3 Hierarchic Graphs . . . 45
3.3.1 Syntax . . . 45
3.3.2 The Multiplicative Model . . . 48
3.3.3 The Additive Model . . . 53
3.3.4 The Time-Extended Additive Model . . . 58
3.3.5 Refinement in the Multiplicative and Additive Models . . . 61
3.4 Architecture Specification – HyACharts . . . 62
3.5 Component Specification – HySCharts . . . 64
3.5.1 The Discrete Part . . . 65
3.5.2 The Analog Part . . . 77
3.5.3 Example: A Typical Hybrid Component . . . 81
3.6 Integration of Other Formalisms . . . 81
3.7 Discussion and Further Work . . . 82
3.7.1 Contribution . . . 82
3.7.2 Related Work . . . 83
3.7.3 Further Work . . . 85

4 Notes on Refinement **87**
4.1 Preliminaries . . . 87
4.2 Architecture Refinement . . . 88
4.3 Behavior Refinement . . . 90
4.4 Discussion and Further Work . . . 93

5 Refinement and Time **95**
5.1 Methodological Aspects . . . 96
5.2 Relaxed HySCharts . . . 100
5.2.1 Constructing Relaxed Invariants . . . 101
5.2.2 Relaxed Analog Dynamics . . . 105
5.2.3 Machine Model for Relaxed HySCharts . . . 107
5.2.4 Remarks on the Relaxation . . . 109

5.2.5 Relaxed HySChart for the EHC's Controller . . . 109
5.3 DiSCharts . . . 112
5.3.1 Machine Model . . . 112
5.3.2 The Discrete Part in Discrete Time . . . 114
5.3.3 The Analog Part in Discrete Time . . . 114
5.3.4 The Discrete-Time Component . . . 118
5.3.5 Interface to Dense Streams . . . 119
5.3.6 DiSChart for the EHC's Controller . . . 120
5.4 Time Refinement of HySCharts . . . 122
5.4.1 Refinement Conditions . . . 122
5.4.2 Sampling for the Discrete Part . . . 126
5.4.3 Sampling for the Analog Part . . . 132
5.4.4 Time Refinement of the EHC's Controller . . . 140
5.4.5 Sampling Rate Validation . . . 143
5.5 Time Refinement in Component Networks . . . 144
5.6 Separate Implementation of the Discrete Part . . . 147
5.7 Discussion and Further Work . . . 149
5.7.1 Contribution . . . 149
5.7.2 Related Work . . . 151
5.7.3 Further Work . . . 155

6 Properties of Hybrid Systems 157
6.1 Example . . . 158
6.2 Systems under Consideration . . . 159
6.3 Properties . . . 161
6.3.1 Robustness . . . 162
6.3.2 Optimality . . . 163
6.3.3 Stability . . . 164
6.3.4 Attraction . . . 168
6.3.5 Further Properties . . . 171
6.3.6 Classification of the Properties and its Consequences . . . 174
6.4 Some Proof Concepts . . . 177

6.4.1 State-based Stability . . . 178
6.4.2 Attraction . . . 183
6.5 Applications . . . 186
6.5.1 Stability and Attraction for the EHC . . . 186
6.5.2 Attraction for Self-Stabilizing Algorithms . . . 190
6.6 Discussion and Further Work . . . 193
6.6.1 Contribution . . . 193
6.6.2 Related Work . . . 195
6.6.3 Further Work . . . 197

7 Summary and Conclusion **199**
7.1 Summary . . . 199
7.2 Conclusion . . . 201
7.3 Further Work . . . 202

A Proofs **205**
A.1 Proofs about HyCharts and DiCharts . . . 205
A.1.1 Totality of HySCharts Revisited . . . 205
A.1.2 Inductive Reasoning for HySCharts . . . 209
A.1.3 Time Guardedness of HySCharts . . . 214
A.2 Discretization as Refinement . . . 215
A.2.1 Operational Semantics of DiSCharts . . . 215
A.2.2 Proofs about Discretization as Refinement . . . 218
A.3 Proofs about Stability and Attraction . . . 223

B Mathematical Foundations **227**
B.1 Sets and Orders . . . 227
B.2 Some Topology . . . 228
B.3 Metric Spaces . . . 231
B.4 Convergence in Metric Spaces . . . 232
B.5 The Metric Space of Streams . . . 232
B.5.1 Divergence Closure and Topological Closure . . . 234

Bibliography **237**

List of Figures **255**

Glossary of Symbols **259**

Index **263**

Chapter 1

Introduction

Today more and more computing elements are used in technical plants and in all kinds of devices. Basic reasons for this tendency are the declining price and size of computing elements accompanied by the possibility of realizing new functionality and by the flexibility software-based solutions offer. Examples range from avionics and automotive applications to automation, process control, telecommunications and consumer electronics. In all such applications, software plays a central role in achieving the functionality of a product. Hence, the development of the software plays a key role in the overall development of the product. A well-founded methodology based on formal methods can assist the systematic development of such *embedded systems* and help to reduce the risk of errors. This is vital for embedded systems as they typically target the mass market or perform safety-critical functions, which means that faults in them are expensive if not unacceptable.

1.1 What are "Hybrid Systems"?

Hybrid (embedded) systems, the focus of this thesis, are a subclass of embedded systems which are characterized by the continuous evolution of the system's variables interrupted by discontinuous changes due to the system logic at isolated time instants. Typically they are heterogeneous and consist of digital and analog components as well as of components that can not clearly be assigned to one of these two worlds at the beginning of the development process. Their mixed digital/analog nature makes hybrid systems an interdisciplinary topic mainly influenced by computer science and control theory. Demanding examples are flight control systems, such as the Airbus A320 fly-by-wire system, which comprises different discrete modes: *take off, cruise, approach* and *go around* [Swe95]. Figure 1.1 depicts the general architecture of hybrid systems. It is explained in detail later in this section.

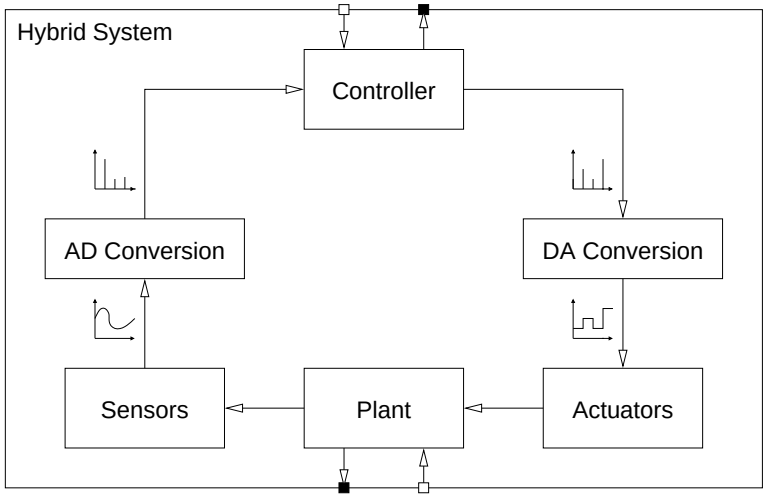


Figure 1.1: General architecture of hybrid systems.

Some terminology. Hybrid systems are often described in the literature as systems with a complex interaction of discrete and continuous dynamics. As the notions *discrete* and *continuous* are ambiguous, we would like to clarify them here.

Control theory distinguishes between continuous-time, discrete-time, and discrete-event systems [Zei76, GW98]. When using the real numbers as time model, continuous-time systems are active over the entire time axis processing their input and producing output. Discrete-time systems can only react to their input and produce new output at distinct, equidistant time instances. Discrete-event systems produce output *events* whenever input events or timeouts occur. Events can typically be regarded as singular signals which only persist for a single time instant. Note that a condition which becomes true because a continuous variable, such as time, reaches a certain value can also be regarded as an event. This in particular implies that timeouts can be seen as events. Unlike discrete-time systems, discrete-event systems are not bound to an equidistant time grid. Instead, they react whenever activated by input or timeout events.

Furthermore, control theory distinguishes between systems operating on quantized (or discrete) values and systems operating on continuous values. Quantization results when the quantities of a physical system are measured and converted for further processing by a digital machine. With this basis, the term *digital system* refers to discrete-time systems with quantized values. If large enough bit vectors are used for quantized values, quantization effects can be regarded as random noise. Therefore, these effects are often neglected in practice and the term digital system is also used for discrete-time systems with continuous values [Vac95]. (Nevertheless, quantization effects are a concern in hardware design if the size of bit vectors is supposed to be minimized.) The term *analog system* refers to continuous-time systems with continuous values. We follow this common, slightly ambiguous terminology. Furthermore,

when the context is clear we refer to discrete-event systems as *discrete systems* and to continuous-time systems as *continuous systems* in this thesis. Systems with quantized values of continuous variables are not considered further.

Hybrid systems in detail. With this terminology at hand we can make the notion of hybrid systems more precise. In our view, there are two aspects which make up the *hybrid* character of a hybrid system. First, there is the interaction of dynamics with different underlying time models. Here, most work focuses on systems with intermixed discrete-event and continuous-time dynamics, with continuous values (cf. the hybrid systems conference series [GNRR93, Ant95, AHS96, HS98, VvS99, LK00, DBSV01]). This is reasonable since discrete-time systems can be seen as a subclass of discrete-event systems which are triggered by timeout events in a specific way.

The second aspect of the character of hybrid systems is the mixing of logical decision making from the computer science domain and of discrete-time or continuous-time control laws as usually considered in the control theory domain. In this thesis, we will refer to this logical decision making as *state-transition logic* because the decisions usually cause qualitative changes in the state of a system. Usually hybrid systems contain both of these aspects—mixing of time models and mixing of computer science and control theory. We illustrate this on the basis of the general architecture of hybrid systems depicted in Figure 1.1.

The controller in the figure operates in discrete-time and is usually implemented in software. Sensors, plant and actuators operate in continuous time. The term *plant* refers to the system to be controlled in control theory. In computer science, the term *environment* is used similarly in the context of embedded systems, but more vaguely in that it refers to all components outside the actual embedded hardware and software. Here we will use the term environment to denote all components of a hybrid system except the controller. The AD conversion component performs an analog-to-digital conversion of its inputs. Typically this involves filtering of the inputs to obtain bandlimited signals, sampling the signals at discrete time instants and holding the sampled value constant in order to convert it to a quantized binary representation that is available at every clock tick as output of the conversion. Analogously, the DA conversion component performs digital-to-analog conversion. This typically consists of converting the quantized binary representation of the present value to a continuous representation and holding this output constant until the next binary signal arrives with the next clock tick. Moreover, the produced piecewise constant output is often passed through a filter to eliminate too high frequencies otherwise present in the output signal. There are also other alternatives to this simple sample-and-hold scheme, which for example involve more complex interpolation in the DA conversion. While these are important for digital signal processing to reduce the cost of required analog filters [Smi97], they usually do not yield better performance of control systems, as Ogata mentions in [Oga87]. Besides this (digital) control loop there may also be a (continuous) control loop around the plant in specific cases. With the interfaces of

controller and plant to the outside world the hybrid system interacts with its surroundings. For instance, if the plant was the engine of a car, the controller would receive input from the driver via the pedal and the engine's output would lead to an acceleration. This in turn would cause the driver to adjust the pedal according to his or her preferences and to the traffic situation.

Obviously, this architecture of hybrid systems involves different time models. However, if there is no state-transition logic part in the dynamics of controller or plant, standard methods from control theory can be used for discrete-time or continuous-time controller design [Föl90, Oga87]. For the hybrid systems community, the system becomes interesting if the controller either decides between employing different control laws (or *control modes*) or the plant involves switching between qualitatively different kinds of continuous behavior. An example of control moding is a situation where a controller switches between a slow, accurate control law for fine adjustment and a fast, less accurate one for approaching a given desired value. An example of switching in the plant is hysteresis or friction effects where, depending on the given forces, stick friction or slip friction occurs. Switching effects in the plant usually result from abstractions when modeling a physical system. There, a modeler may decide that a certain effect takes a very small amount of time relative to the behavior he is interested in and that it may therefore be assumed to happen instantaneously [Mos97]. Finally, note that the AD and DA converters and the filtering and processing they may involve are inherently mixed analog/digital systems. This justifies that the techniques for the development and analysis of hybrid (control) systems are also interesting for the field of analog/digital codesign [DSG+94, GW98], and vice versa.

Example Systems. Besides the avionics domain referenced in the beginning of this section, interesting examples of hybrid systems can be found in automotive electronics. These include simple systems, such as the automatic transmission control provided as example in [TMI99], as well as complex products, such as autonomous cruise control systems (ACC). These ACC systems adjust a car's speed and distance to the car in front depending on different traffic situations, e.g., highway or stop-and-go traffic [Dai01]. In particular, the ACC system demonstrates how new functionality is obtained by using information technology to couple hitherto autonomous subsystems such as automatic transmission control, distance sensors, braking systems and, in the future, probably mobile communication as well. Further examples include walking robots, where different discrete phases follow one another and each one is characterized by different continuous dynamics [BGM93], or manufacturing plants. For instance, in [PPS00] a wire production plant is regarded, where different control paradigms are employed in different phases of the process of coiling up wire.

Methods for hybrid systems. In contrast to ordinary discrete or continuous systems, the development of hybrid systems requires *integrated* methods. These methods must take all aspects of a system into account such that the overall system as given in Figure 1.1 satisfies the desired properties affecting discrete *and* continuous dynamics.

Needed methods include modeling, simulation, verification and synthesis techniques. Since discrete-time models can be regarded as a special case of discrete-event models, inventing such techniques based on a model incorporating discrete-event and continuous dynamics is reasonable.

1.2 Subject of the Thesis

The aim of this thesis is twofold. First, it proposes a systematic approach to the development of hybrid systems together with some infrastructure. Second, it intends to foster a deeper understanding of hybrid systems by computer scientists.

For the development of hybrid systems, the thesis suggests a development process which aims at helping to reduce design risks and gaining flexibility. This is achieved by postponing the important step of partitioning a hybrid system in discrete-time and continuous-time subsystems to the late development phases, when enough confidence in the design has been obtained. A prerequisite for such a process is the availability of description techniques and validation techniques for hybrid systems. Furthermore, transformations are needed which permit refinement of a hybrid system and moving from an abstract hybrid model towards an implementation which contains subsystems operating in discrete time.

As a contribution to these prerequisites, the thesis defines precise, visual description techniques for the data flow and the control flow in hybrid systems. These techniques can be coupled with description techniques for discrete-time systems, and they are close to techniques commonly used in software engineering, like UML [Gro00] and Statecharts [Har87]. An integration of block diagrams, as used in control theory [PH88], is straightforward.

For the introduced description techniques, property preserving transformations, also called *refinement methods*, are discussed. The focus of these transformations is on enabling the migration from system components which operate in a continuous time scale to components operating in discrete time. Such transformations close the gap between existing techniques for hybrid systems and discrete-time systems. The thesis examines effects arising from a discrete-time implementation and develops general conditions which, when satisfied, provide that moving to discrete-time preserves essential properties of the original system. Above that, methods are presented which support to design models that satisfy the identified conditions. While the results are given for the specific description techniques used in this thesis, they can also be carried over to related notations which follow the style of hybrid automata [ACH+95].

The formal notion of refinement, which is used in this thesis as basis for the transformations, is motivated by a general study of the properties of hybrid systems. These properties are formalized, classified and related to refinement. The thesis also proposes proof techniques for some of the properties and explains why some of them have

not been of interest to computer science so far. Particular emphasis is put on formalizing parallels between computer science and control theory which become apparent in the definition of the properties and proof methods. This helps to advance a better understanding of hybrid systems.

In order to illustrate how the methods proposed in this thesis are applied, they are used to elaborate various aspects of a non-trivial example system.

1.3 Structure of the Thesis

The thesis is structured as follows. After some notes on the development process of control systems, Chapter 2 proposes an integrated development process which is characterized by the late partitioning of a hybrid system into continuous-time and discrete-time subsystems. The proposed process is contrasted to a development process with an early partitioning, and related work supporting the integrated development process is discussed.

In Chapter 3, the hybrid description techniques *HyCharts* are introduced, and hierarchic graphs, which are the semantic foundation for HyCharts, are defined. HyCharts come in two flavors, *HyACharts* for the specification of the data-flow (or the architecture) in hybrid systems and *HySCharts* for the specification of the control-flow (or the behavior) of a hybrid system's components.

General notes on refinement rules which do not affect the time model of HyCharts are given in Chapter 4. The transferability of known refinement rules for the data-flow and the control-flow in discrete systems is briefly discussed there.

In a central theorem, Chapter 5 defines under which conditions a discrete-time component is a refinement of a HySChart. As preliminaries for this theorem, the effects of a discrete-time implementation are discussed and a liberal variant of HySCharts is introduced. For the specification of discrete-time components, *DiSCharts* are defined as a discrete-time variant of HySCharts. Furthermore, methods are developed which help to systematically construct a discrete-time refinement of a HySChart. These methods also involve the usage of techniques from numerical mathematics and control theory.

Chapter 6 studies and formalizes general properties of hybrid systems as the underpinning for the used refinement notion. The properties are classified and the effects of refinement on them are discussed. Due to their importance, the chapter introduces proof methods for some of the properties. The properties and proof methods are compared with computer science and parallels are formalized. This provides a deeper insight into hybrid systems and, in particular, it identifies the vital role of topology in the properties of hybrid systems.

Each chapter ends with a discussion of the results, an overview of related work and directions for future work. Chapter 7 presents some general conclusions.

The Appendices provide the necessary foundations for results in this thesis. Appendix A formalizes some basics which are needed for proofs concerning HySCharts and DiSCharts, including their operational semantics. Furthermore, proofs for those theorems are given which do not occur in the main part of the thesis. Appendix B introduces foundations of (partial) order theory, topology and (complete) metric spaces.

Figure 1.2 depicts the dependencies between the chapters of this thesis. The appendices are omitted in the figure.

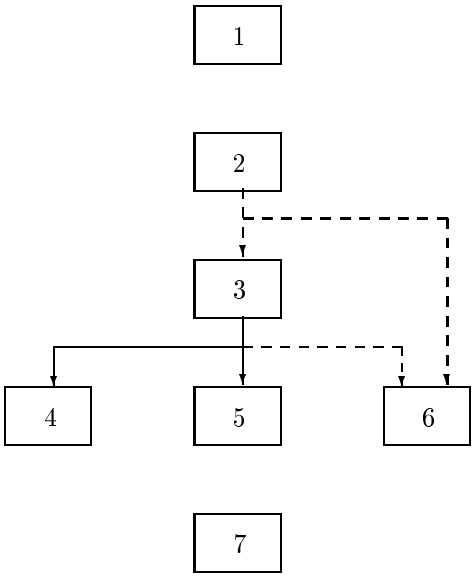


Figure 1.2: Chapter dependencies (numbers refer to the chapters).

1.4 The Running Example

We use an *electronic height control system (EHC)* as a running example in the thesis. All fundamental concepts which we present will be outlined along this example. The purpose of this system, which was originally proposed by BMW, is to control the chassis level of an automobile with pneumatic suspension. The abstract model of this system, which considers only one wheel, was first presented in [SMF97]. The versions we consider here are derived from the presentation in [GSB98a]. The example is introduced in detail in Section 3.1. Here, we explain in which succession the models presented throughout the thesis would occur in a development process.

The most abstract version of the EHC if given in Chapter 6. Among the models given in the thesis, this one would occur first in a development process. Some initial

properties would be established for it, as in Chapter 6. In further development steps, this model would be split into components until the system architecture presented in Chapter 3 is obtained. Note that we do not consider these development steps in the thesis. For one component of the system, which will be called *Control*, the behavior could then be specified by the HySChart in Chapter 5. The development step explained in that chapter can then be used to refine this component by constructing a discrete-time model for it. This is the final model of the component *Control* we consider in the thesis.

In Chapter 3, a further model of the behavior of this component is presented. This model is not supposed to occur in a development process which follows the paradigms introduced in Chapter 2 because it mixes discrete-event and discrete-time aspects within one component. This mixing is motivated by the purpose of the model in Chapter 3. There, it is not intended to demonstrate good modeling practice, but it instead serves to explain almost all characteristics of the HySChart formalism.

Chapter 2

Remarks on the Methodology for Hybrid Systems Development

This chapter describes how the development of hybrid systems can benefit from notations, methods and tools that provide an integrated view on a system's discrete and continuous, computer science and control theory aspects. As a prerequisite, Section 2.1 explains the development of control systems and indicates the role synthesis techniques play there and when they can be applied. Section 2.2 first outlines characteristics of the development of hybrid systems in practice these days. Then, a process is presented which aims at bringing more mathematical rigor and an integrated view on the different aspects of hybrid systems to their development. The process relies on integrated formal notations, analysis and refinement techniques. The remaining chapters of this thesis are classified w.r.t. their contribution to the proposed integrated development process. Furthermore, we explain how related work supports the integrated process. Note that the chapter focuses on analysis and design phase. A preliminary version of the ideas presented here appeared in [PSS00].

2.1 Preliminaries: Development of Control System

In order to understand the development of hybrid systems from a computer science point of view, it is reasonable to get acquainted with the basic development steps in the design of control systems. We therefore describe this process in this section and compare it to the software development process.

2.1.1 Development Steps

The usual control problem consists of designing a controlling device *Controller* which controls a given physical process *Plant* such that specific quantities of the process have

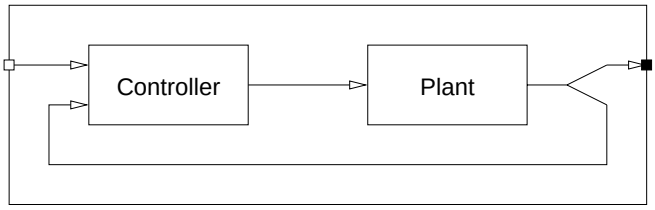


Figure 2.1: General structure of a closed-loop control system.

certain desired values which are input to the controlling device (Figure 2.1). The user interface is not considered explicitly in the figure. It may either be part of the controller or it may be external and provide input to the controller. According to [PH88], typical requirements for the controlling device are compensation of disturbances, compliance with acceptable deviations from ideal behavior during and after disturbances and a reasonable degree of insensitivity w.r.t. the parameters of the physical process.

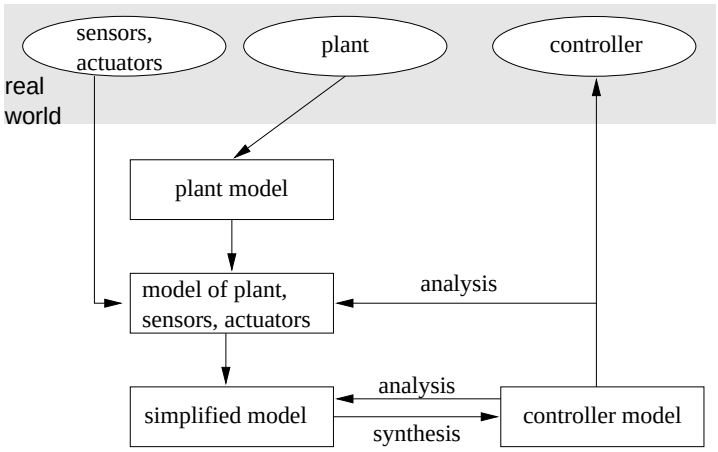


Figure 2.2: Development of control systems.

Developing such controllers ideally consists of the following steps [PH88, Toe96] (see Figure 2.2). First, a mathematical model of the physical process, the plant, is set up, validated w.r.t. the real system (e.g., by experiments and simulation) and analyzed. Sensors and actuators are chosen based on the knowledge about control problem and plant. Then, they are also modeled mathematically and validated. The obtained mathematical model of the plant including sensors and actuators is used to design the controller. If the mathematical models are sufficiently simple, algorithms can be used to automatically produce a mathematical model of a controller which is optimal w.r.t. certain given criteria (e.g., by solving a Ricatti equation [Fri86]). Otherwise, the models are simplified until automatic techniques become applicable or a manual design is employed. Thereafter, the model of the overall system is analyzed analytically or by simulation. In succeeding steps, the simplified model parts are replaced by more

accurate models or by hardware-in-the-loop[1] and the system is verified with further simulations. Finally, the controller model must be mapped to the real world and tested in interaction with the remainder of the system. If any of the verification steps fail, the respective design steps must be iterated. In contrast to conventional control engineering, mechatronics also considers how the physical system can be modified to improve performance of the overall system. Standard methods exist for the controller design of *linear time invariant* systems. A system is linear if the superposition principle applies, i.e. if y_1 is the system response to input x_1 and y_2 is the response to x_2 then the system is linear iff input $ax_1 + bx_2$ results in $ay_1 + by_2$. It is time invariant if the system is independent from absolute time, i.e. if time shifts of the input result in the same time shifts of the output. Linear differential equations with constant coefficients are a standard way for describing linear time invariant systems (see [Föl90, Fri86]). Model simplifications therefore often consist of replacing non-linear differential equations by linear ones around some given parameters. The mathematical background here is Taylor expansion [Kön90]. Note that simplifications in this context are also called *abstractions*, although they do not always have the formal background which is often associated with the term in computer science. Furthermore, note that the state-transition logic in hybrid systems is a source of severe non-linearity.

The design of digital controllers of analog plants is based on techniques which are essentially similar to those for analog controllers. Here, linear difference equations take the place of linear differential equations. Shannon's sampling theorem, which implies that a continuous signal with a given highest frequency f is completely determined by sampling its value with (at least) frequency $2 \cdot f$ [Uns00], enables one to build meaningful models of analog plants by means of difference equations.[2]

The presence of powerful controller synthesis techniques stresses the importance and the benefit of formally modeling the plant. In computer science, process models such as the V-model [Bal98] or Catalysis [DW98] also require that environment models be set up. However, in practice such models often remain limited to class or architecture diagrams and do not consider system behavior in greater detail. From the author's point of view, the development of embedded systems in particular could greatly benefit from more formal environment models. Even if synthesis is not possible from the models, they are needed for thorough requirements specification and for formal verification, as well as for testing.

Actual controller design may differ from the steps outlined above. As [PH88] indicates, in practice standard controller components such as PID (proportional-plus-integral-plus-derivative) controllers are often selected and simply parameterized appropriately

[1] Hardware-in-the-loop means that in the model of a system under development one or more model components are replaced by the corresponding real components and connected ("in the loop") to the rest of the model for evaluation in real time. For instance, in the case of the design of an engine controller, the real engine may be connected to the rest of the model which still only exists on a prototyping platform.

[2] In practice a frequency of 8 to 10 times f is often used for digital control of good quality [Oga87].

by means of simulations.

Example 2.1 (State-space representation, controller synthesis.) As an example, we want to describe the state-space representation of linear time invariant systems and want to indicate the potential of mathematical methods for them. Formally, a (continuous-time) system is in *state-space representation* if it is defined by two matrix (differential) equations of the following form [Föl90]:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where x is the state vector, u the input vector and y the output vector of the system. A, B, C and D are constant matrices, $\dot{x}$ denotes the time derivative of x and t denotes (continuous) time. Vector x models all those quantities of the physical system whose values entirely determine the future behavior of the system. As the state vector is often not directly measurable, output vector y models the externally visible behavior of the system.

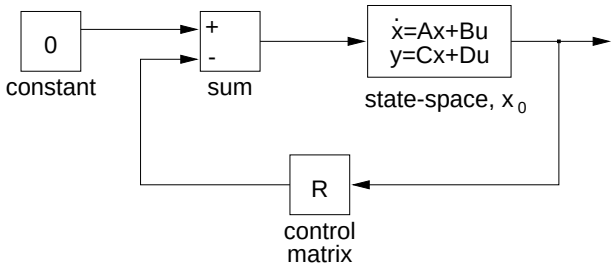


Figure 2.3: Block diagram for the control of a linear time invariant plant in state-space representation.

To demonstrate the use of mathematical methods, we assume a plant in state-space representation to be given and propose a control law which drives x to 0. For simplicity let C be the identity matrix and D the 0 matrix such that $y(t) = x(t)$ holds in our example. In order to drive x to 0 we use $u(t) = -Ry(t)$ as control law, where R is a further constant matrix. The system consisting of controller and plant is depicted in Figure 2.3 as a *block diagram*. Block diagrams are explained in greater detail below. The behavior of the overall system is given by $\dot{x}(t) = (A - BR)x(t)$. Solving the differential equation for initial value x_0 yields $x(t) = x_0\, e^{(A-BR)t}$, where $e^{(A-BR)t}$ is a notational shorthand for the exponential series with matrix $A - BR$. The Eigenvalues of $A - BR$ are crucial for the behavior of the system. They determine if and how the system diverges, oscillates or converges to 0 for an initial value different from 0. In control theory, methods exist for determining matrix R from A and B such that $A - BR$ has given Eigenvalues, provided A and B satisfy certain restrictions. This is called *pole assignment* [Won67]. □

2.1.2 Important Notations

Block diagrams. Control theory predominantly uses *block diagrams* to structure and graphically describe models. Figure 2.3 is an example. From a computer science point of view, block diagrams can be regarded as data-flow graphs that are built up from predefined components. Every block of a block diagram represents a functional relationship between its input variables i, symbolized by incoming arrows, and its output variables o, depicted as outgoing arrows. Usually this functional relationship is specified by labelling the block with one of the following: the relationship itself, a symbol, like $\int$, which denotes the relationship, a characteristic curve, a differential equation or a *transfer function,* which defines the relationship. Transfer functions result from the Laplace transformation of differential equations and are widely used in control theory, because they simplify manual analysis of control systems and specification of filters. Computer-based analysis favors state-space representations instead. Laplace transformation is briefly described in [PH88]. Given the functional relationship f of a block, it semantics is given by $o = f(\iota)$ for input signal ι and output signal o. The coupling of blocks is as in usual data-flow graphs. We do not go into further detail here, since Chapter 3 defines the semantics of a notation for data flow. The techniques used there can easily be carried over to block diagrams.

In the case of block diagrams for discrete-time systems, difference equations are used instead of differential equations and transfer functions result from z-transformation, which is the discrete-time counterpart to Laplace-transformation [PH88]. Computer-aided engineering (CAE) tools like MATLAB/Simulink [TMI00] offer rich libraries for block diagram descriptions. Such tools also allow the usage of AD and DA converters for coupling blocks with different underlying time models and they even allow the coupling of discrete-time blocks with different sampling rates. The semantics of the resulting diagrams can be greatly obscured in the presence of such incompatible time grids or in presence of library components which are not mathematically defined but use, for example, C code instead. Thus, while block diagrams in their simple form can be regarded as graphical descriptions of mathematical equations, this is not true for the complex diagrams often drawn with CAE tools. These correspond to (graphical) progamming languages rather than to precise equations.

Bond graphs. From a mathematical point of view, *bond graphs* are a helpful alternative notation for modeling physical systems [Pay61, Mos97]. In essence, bond graphs try to put more of the underlying physics into the visual formalism than block diagrams do. The guiding principle behind them is energy preservation and the transfer of energy within a system. They associate *effort* and *flow variables* with all arcs, called *bonds*, in the graph. For components in electrical circuits, for instance, voltage is the effort variable and current is the flow variable. For each node, its influence on effort and flow variables is specified. The formalism ensures that domain-independent generalizations of Kirchhoff's laws hold at all junctions of bonds and that energy preservation

holds for the overall system. Due to their support for basic physical principles, bond graphs are well suited for modeling the physical world. The direction of arcs in bond graphs merely visualizes the sign of energy transfer; it does not reflect computational causality, which is beneficial in the description of physical components like gearboxes that can transmit energy in both directions. In contrast, block diagrams do associate causality with the direction of arcs, although this causality is not present in the underlying mathematical equations. Instead, it results from the engineers cause/effect interpretation which is adequate for controllers. As a result, bond graphs do not have great relevance in control theory. However, in mechanical engineering they do have at least some relevance, in particular for environment modeling. This is underlined, for example, by the comprehensive libraries for mechanical engineering which exist in the modeling language standard *Modelica*, which is in turn based on the bond graph paradigm [MOE99].

Example 2.2 (Bond graph for an RLC circuit.) An example of a bond graph is depicted in Figure 2.4. The example is taken from [Bro99a]. The positions labeled S_e,

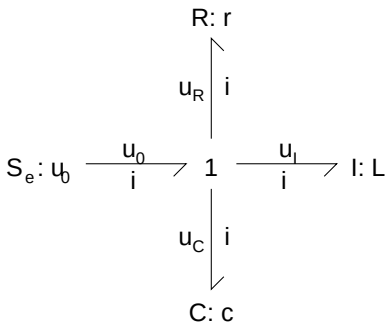


Figure 2.4: Example bond graph.

R, I and C refer to predefined elements of the bond graph formalism. The constants u_0, r, L and c juxtaposed with them specify specific parameters in the equations which are associated with those predefined elements. Effort, denoted by label u in the figure, and flow, denoted by label i, are associated with each arc (or bond) in the figure. The *junction* labeled 1 defines a specific coupling of the efforts and flows which are associated with the bonds connected to the junction. For the junction in the figure, this coupling is Kirchoff's voltage law. It implies that the flows for all arcs are equal. The equations corresponding to the depicted bond graph are:

$$\begin{aligned} u_R &= i \cdot r && \text{(position } R\text{)} \\ u_I &= L \tfrac{di}{dt} && \text{(position } I\text{)} \\ u_C &= \tfrac{1}{c} \int i\, dt && \text{(position } C\text{)} \\ u_0 &= u_R + u_C + u_L && \text{(Kirchoff)} \end{aligned}$$

These equations, for example, describe the dynamics in an electrical RLC circuit, which consists of the sequential composition of a voltage source S_e, a resistor R, an inductivity I and a capacity C. □

2.2 Hybrid Systems' Development

With this background concerning the development of control systems at hand, this section discusses the development process for hybrid systems and outlines how it can benefit from *integrated* techniques which consider discrete as well as continuous aspects. In our discussion we concentrate on the analysis and design phases. This is justified by the great importance of the documents produced in the analysis phase. In [Bro97c], Broy explains that according to an investigation in industry, 50% of the problems which occur within delivered embedded systems and which are reported by the customers are caused by misconceptions in capturing the requirements. Furthermore, we think that the early phases of hybrid systems' development bear most potential for improvement. Once a detailed design of the system under development is present, coding could be performed essentially automatically by appropriate CASE tools, provided control algorithms are adequate for the underlying problem and performance optimization of the produced code is not a major issue. For example, hardware and software infrastructure supporting automatic implementation of prototypes is developed in [FKMF97, PMK+99, MF00].

Hybrid systems often perform safety critical tasks or target the mass market. To reduce the risk of highly expensive errors in them, we advocate the use of formal methods and notations wherever possible in their development. *Formal* here means that the methods and notations have a well-defined mathematical foundation and semantics. Unambiguous specifications are typically required by standards for the development of safety critical systems, such as DO-178B [RTC92] in the avionics domain. Notations with formal semantics can help in obtaining this goal because of their mathematically precise meaning.[3] Further motivation for the application of formal methods stems from the rigorous validation and verification techniques they enable, from the possibility to precisely define the interaction between different formalisms and from a mathematically guaranteed requirements traceability which can be obtained if formal methods are applied consequently. We will consider these general aspects in the context of hybrid systems' development below.

Nevertheless, not all requirements for an intended system can be documented with clearly defined semantics. In this thesis we focus on functional, performance and time requirements. We do not consider other classes of requirements, such as fault-tolerance requirements and requirements resulting from the selection of specific hardware or

[3] Nevertheless, this meaning can differ from what a designer thinks the (formal) document expresses.

operating system platforms (e.g., resource limitations and scheduling aspects), unless they can be expressed in a functional manner. Non-functional requirements are not directly affected by the methods proposed in the thesis and are therefore not considered further. Note that we also regard detailed specifications as functional requirements on a low implementation-oriented level.

In the following, we propose an integrated development process for hybrid systems and contrast it to a conventional process. The integrated process results from carrying over ideas such as graphical specification with different systems views and model-based validation with formal methods to hybrid systems. A central characteristic of the proposed approach is that it is based on notations which have clearly defined semantics. The basic idea in the proposed process is to push the point at which development based on formal methods can start towards the beginning of the analysis phase. The resulting earlier availability of powerful formal methods-based validation and verification techniques, such as model checking, helps to reduce design risk. Section 2.2.3 explains how the results of the succeeding chapters of this thesis contribute to the feasibility of the proposed process. Although a seamless formal methods-based development of hybrid systems is unrealistic today and will possibly never be feasible in practice, the study of elements of such a process is nevertheless highly valuable in at least two respects. First, it can result in increased automation or formal guidance of *some* steps in the development of hybrid systems, like, e.g., the selection of sampling rates for discrete-time components. Second, it helps to obtain a deeper understanding of the immanent problems in the development of hybrid systems. This understanding in turn can help to solve specific problems occurring in practice.

2.2.1 A Conventional Development Process

In a conventional development process, the notations which are available for requirements documentation usually are informal text, some kind of automata and discrete-time and continuous-time block diagrams or architecture diagrams. In industry, tool couplings, such as Statemate coupled with Matrix_X or the MATLAB/Simulink/StateFlow environment [FEM+98, CWM98], are popular. While such tools allow the documentation of the detailed design of a system with notations which at least have a formal syntax, they do not offer notations other than informal text to document more abstract system requirements which affect both the discrete state-transition logic as well as ongoing control tasks (Figure 2.5, top). Such requirements typically define the succession of different operation phases, like take-off, normal flight and landing for aircraft, and constraints on the continuous variables and their evolution in these phases. Such constraints may be desired ranges for variables or their derivatives and stability or robustness requirements.[4] In general, control laws are not fixed yet. For

[4] Chapter 6 introduces classes of properties which typically are of interest for hybrid systems and also explains notions like stability and robustness in detail.

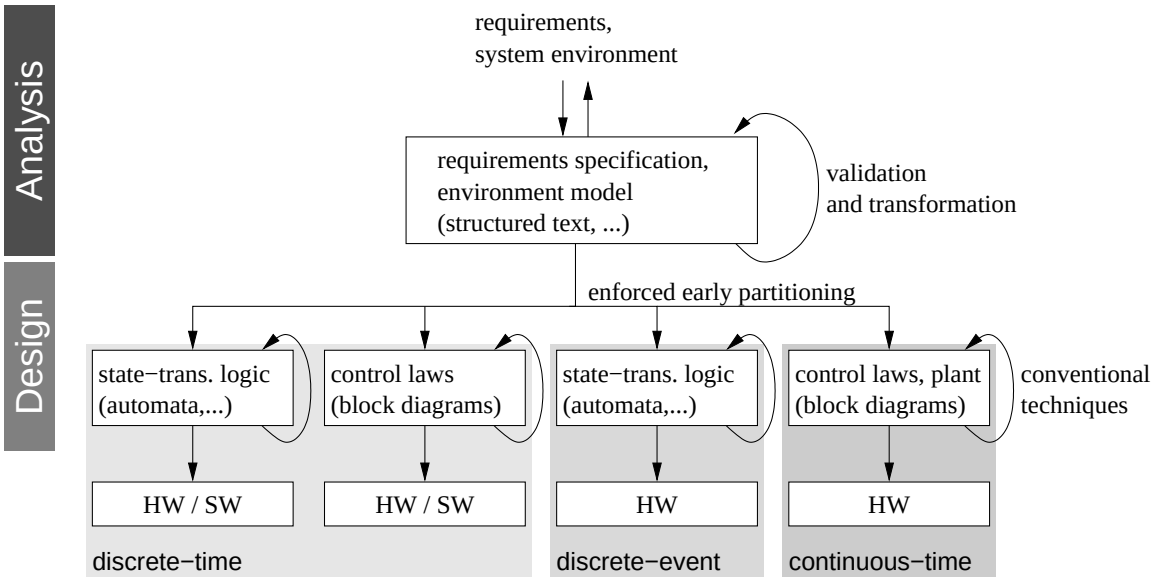


Figure 2.5: A conventional development process, based on isolated description techniques (indicated in parenthesis).

such constraints the real axis as time model is adequate, since in the physical world the values of these quantities are *always* relevant, not only at single time instants. For digital controllers of continuous plants this corresponds to the fact that the system's behavior between sampling instants, i.e. its *intersample response*, must also be acceptable w.r.t. the desired aim. For instance, a situation in which the system exhibits oscillations that are not detected with sampling is undesirable. Models with the real numbers as time model can be regarded as abstract, based on an arbitrarily fine time grid. For such models, moving towards implementation includes fixing a specific time granularity. Furthermore, we remark that in order to be able to speak about derivatives and changes in variables over time, a quantitative model of time is mandatory.

A second problem in the conventional process results from the need to also elaborate a model of the environment in the analysis phase. Such a model is necessary to document assumptions about the environment and to serve as testbed for preliminary versions of the controller. Moreover, it is a basis for synthesis techniques, if existent. Continuous-time block diagrams or differential equations can well be used for environment modeling, if the environment merely exhibits continuous-time behavior. However, if there also are complex discrete changes in the otherwise continuous environmental dynamics, using block diagrams with switches or systems of switched differential equations soon becomes difficult to survey. In tools such as MATLAB/Simulink/StateFlow, this dynamics can be expressed with an extended state machine whose behavior depends on the continuous-time block diagram and which exports its state to the block diagram. There, a switch selects the sub-diagram for output which is required to be

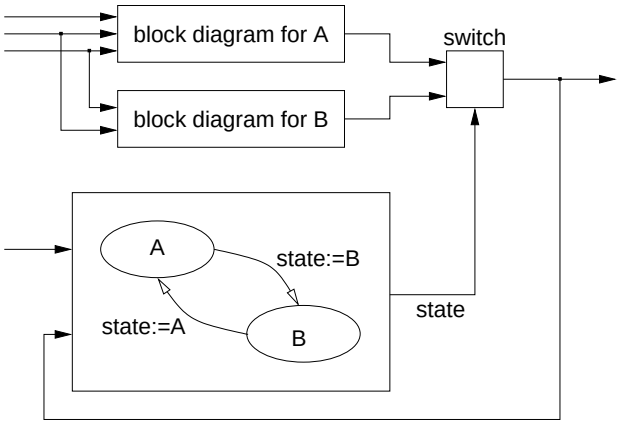


Figure 2.6: Coupling of a state machine and block diagrams in tools such as MATLAB/Simulink/StateFlow.

active in the respective state of the machine. This way the correspondence between the states and the sub-diagrams is obscured. Figure 2.6 depicts such a model schematically. In practice, there are sometimes legacy simulation models, e.g., in FORTRAN code, simulating the mixed discrete and continuous behavior. In all these cases, the underlying notations lead to an environment model which is unnecessarily difficult to understand.[5] This situation gets worse, as there is no clearly defined semantics for the coupled notations, unless mathematical formulas are used. These, however, are difficult to understand, particularly in the case of bigger systems.

Besides these modeling problems, a designer has to perform a number of development steps informally in such a development process (i.e. without documenting them with clearly defined notations) before a properly documented process can start (i.e. a process relying on formal description techniques). The arrows at the top and in the middle of Figure 2.5 indicate these steps. They include a partitioning step which consists of two parts: First, requirements which only affect the state-transition logic and requirements which only affect the control laws have to be derived from the given (hybrid) requirements. Second, these requirements have to be refined to low-level requirements which can be documented with the isolated description techniques for the respective part. The refinement steps may involve implicit changes in the time model. For the state-transition logic in particular, an (implicit) time-discretization of some (discrete-event) parts may occur, which can have unintended consequences. The correctness and appropriateness of the partitioning and refinement steps is difficult to ensure because (1) validation and verification is limited to informal methods such as reviews for these steps and (2) no notations with precise semantics are used which in turn makes reviews more difficult. Simulation as a powerful technique for validation and verification is not possible until the partitioning and refinements steps have been

[5] We do not suggest redesigning such legacy models from scratch, but instead we advocate the use of better suited notations for future models.

performed since simulation tools require at least a formal syntax, which in turn is only present for the partitioned, detailed models in such a development process. However, even when simulation is possible, its results are difficult to interpret since the semantics of the coupling of simulation tools is not precisely defined at least as far as commercial tools are concerned. As a result, such a conventional development process enforces a partitioning of the hybrid requirements because of the lack of adequate integrated formal notations and the lack of validation and verification techniques for them. As in any development process, the late availability of powerful validation and verification techniques in the development process is critical as decisions made in the beginning may turn out to be inadequate later and necessitate a redesign of the whole model. For those system components for which analog and digital implementation alternatives exist, the enforced partitioning is particularly unsatisfactory since it impedes an analog/digital codesign approach for these components and the partitioning decisions may be difficult to alter later on.

Nevertheless, the conventional process also has advantages. When the system under development has been partitioned into state-transition logic and control laws, such that it can be described with the notations listed above, well-known techniques from computer science and control theory can then be applied to the respective parts of the model (see Figure 2.5, arrows at the bottom). For instance, model checking and automatic test-case generation may be used for the state-transition part and analysis of Eigenvalues and controller synthesis for the control part. In this case in particular, we can also use precise notations which have a clearly defined semantics, such as the automata and architecture diagrams supported by a formal tool like AutoFocus/QUEST [Slo98]. Note that we also regard block diagram descriptions as formal here, provided all blocks in them are based on the same time model and, in the case of discrete-time block diagrams, on the same sampling rate. Under this condition, a mathematical model can be associated with individual blocks and their interconnection in a straightforward manner.[6] Yet, the interaction of components developed with the isolated techniques remains imprecise. Usually implicit assumptions about the state-transition logic's behavior are used in controller design. Similarly, informal abstractions of the control part's behavior motivate design decisions for the state-transition logic. The integrated development process outlined in the following section attempts to avoid these drawbacks.

2.2.2 An Integrated Development Process

In a development process with formal hybrid description techniques, such as the one depicted in Figure 2.7, the designer is able to formally specify models for mixed discrete and continuous requirements at early stages of the development process, thereby

[6] Nevertheless, the user has to keep in mind that the selection of integration algorithms for simulation can have a great impact on simulation results and can cause them to differ strongly from the mathematical model.

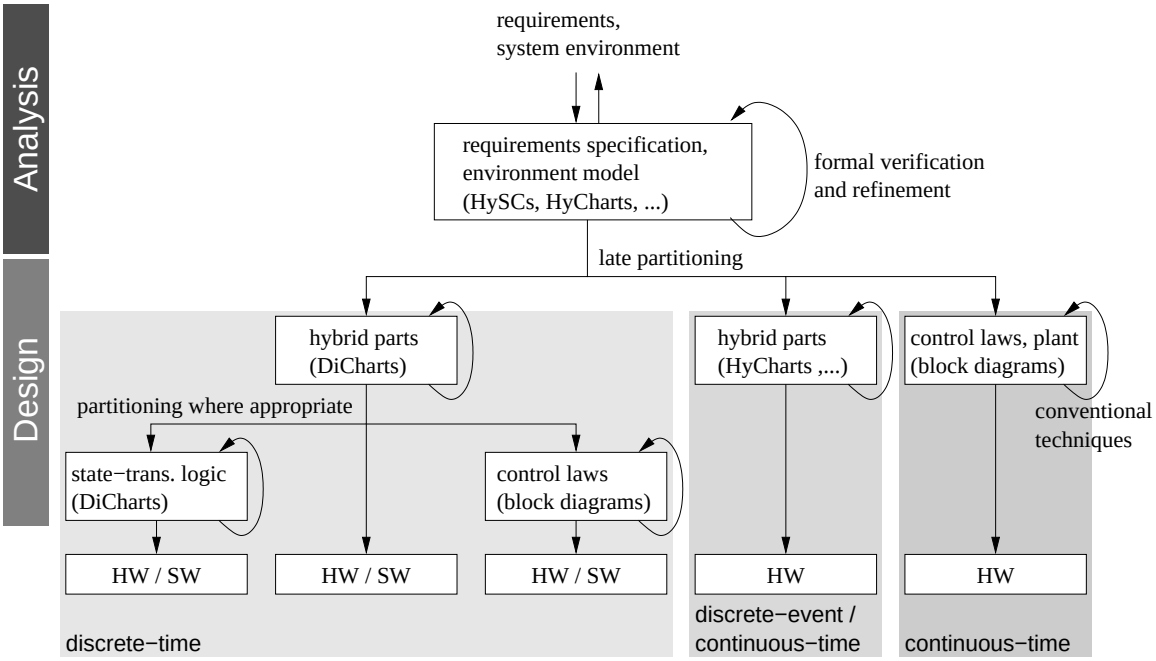


Figure 2.7: Integrated development process, based on hybrid description techniques (indicated in parenthesis). Note that pure discrete-event parts are subsumed in the hybrid parts.

getting precise documentation. If techniques such as simulation, model checking and consistency checking exist for validation and verification of the requirements captured in such an initial model, design risks can be greatly reduced as early as in the analysis phase and the model can be *systematically* designed to reflect the hybrid requirements (Figure 2.7, top). Rudimentary versions of such techniques already exist and are an area of current research (e.g., [HHWT95, DFG01]). In later steps, when enough confidence in the model has been obtained, it can be gradually refined towards the detailed design. If mathematical techniques which ensure that the systems' functionality is augmented without violating previously established requirements are used for this stepwise refinement, the traceability of the previously established requirements is automatically guaranteed. In the context of formal methods we also call this *formal refinement*, or simply refinement when the context is clear. For some model components, refinement towards detailed design involves partitioning them into discrete-time, continuous-time and mixed discrete-event/continuous-time subcomponents, or partitioning state-transition logic and control algorithms into different subcomponents (Figure 2.7, bottom). Typically, the environment model, analog filters and AD and DA converters will be isolated into continuous-time or hybrid componentes. Additionaly, some fast control laws may also be implemented in continuous time. The state-transition logic and most control laws will be implemented digitally, and are therefore specified with an underlying discrete-time execution model in the low-level

requirements.[7] Here, the state-transition logic and control laws may be split into isolated components. However, as long as there are no specific needs enforcing this separation, such as the application of specialized methods, they can remain in a single component as well (see the branches to discrete-time code in Figure 2.7, left). Digital implementation is motivated by the benefits software-based solution offer. Provided computing hardware is present, the same hardware can be used to run software fulfilling a variety of different tasks in parallel. Replication costs for software virtually do not exist. Furthermore, additional system requirements and permorfance optimization can be realized with software updates without necessarily requiring modified hardware.

For the state-transition logic, the step from high-level requirements towards implementation-oriented, low-level requirements also includes a change in the time model, from an initial discrete-event model disregarding implementation specific sampling effects to a discrete-time model as present in digital computers. As in the conventional development process, this transition may introduce undesireable effects caused, for example, by missing some events due to inappropriate sampling. In this process, however, formal methods, like (formal) refinement techniques, can be used to ensure that the transformation maintains desirable properties. In order to obtain discrete-time control laws, standard techniques can be used to synthesize discrete-time controllers of continuous plants which yield the desired dynamics of the overall system. However, if the coupling of state-transition logic and control laws leads to control moding, the behavior of the coupled system has to be analyzed in an integrated manner considering both aspects, since the switching can cause instability or poor performance, despite locally optimal control laws in each mode. One technique supporting such an integrated analysis is [Bra94] where a technique for stability proofs of hybrid systems is given. In our view, splitting state-transition logic and control laws into isolated components should be avoided in presence of control moding, because it entails the danger of focusing on control laws while disregarding the effect of mode changes initiated by the state-transition logic. If a partitioning is required, e.g., for implementation purposes, the acceptability of the behavior of the coupled system should be ensured before the partitioning, because after the partitioning it would be necessary to reassemble the parts for effective analysis, which may be difficult. Nevertheless, it is important to bear in mind that integrated analysis and synthesis methods are still a topic of research. However, the integrated development process at least enables formalization of the problem, while in the conventional process there is no integrated view on state-transition logic and control laws. Thus, clearer documentation is obtained which thereby helps designers to focus on the problem. Besides that, if partitioned submodels are constructed, conventional techniques can still be applied to realize those aspects which only affect the respective part, just as in a conventional development process

[7] Note that even if the state-transition logic builds upon the event mechanism of an underlying operating system, it can nevertheless only react in dependence of the clock rate given by the digital hardware.

(Figure 2.7, bottom).

To summarize the previous paragraphs, the availability of formal hybrid description techniques and supporting methods for them pushes the point at which systematic development can begin to the beginning of the analysis phase. (Systematic development here means development with mathematically precise documentation.) For those system components which can be implemented in a digital or analog manner, a partitioning into discrete-time and continuous-time submodels can be postponed to subsequent development phases. A separation of state-transition logic and (discrete-time) control laws can even be avoided completely. In any case, a development process with hybrid description techniques allows us to obtain greater confidence in the model before a partitioning. Namely, testing and model-checking techniques can be used to analyze requirements and formal refinement techniques can be used to guarantee the traceability of these requirements. By postponing implementation-related questions, changing requirements can be taken into account more easily. Thus, errors made in the initial development phases can be found earlier which in turn makes them cheaper to correct.

Notations. The development process we propose in Figure 2.7 is based on description techniques developed by the author in joint work with colleagues at the Technische Universität München in the last years. For requirements specification and environment modeling, it uses the sequence chart/MSC-like notation *HySC* [GKS00], and the combination of architecture diagrams and a hybrid automata variant which is subsumed in *HyCharts* (Chapter 3, [GSB98a]). Sequence charts and MSC are defined in [Gro00] and [IT96], respectively. In our view, sequence chart-based description techniques, like HySCs, are a valuable complement to architecture diagrams at the beginning of the development process for hybrid systems. In contrast to automata, they are (usually) not interpreted as describing the exact behavior of the system, but rather only some required fragments of it. [Krü00] calls this an existential interpretation of sequence charts. Furthermore, as they typically depict interaction between various components, they help to identify states of the individual components as well as common states of a set of interconnected components. This is helpful for a better understanding of the system as well as for designing automata for the specification of the components' detailed behavior. A methodological transition from HySCs to HyCharts is ongoing work. For similar work on discrete systems see [Krü00]. Succeeding steps in Figure 2.7 refer to HyCharts rather than to HySCs. As notations for the discrete-time part, we propose discrete-time block diagrams and DiCharts, a discrete-time variant of HyCharts in which state activities can be used for the specification of state specific discrete-time control laws. For the continuous-time part we suggest (continuous-time) block diagrams. Block diagrams can be integrated easily into HyCharts and DiCharts.

Tool support. While there is hardly any tool support for the integrated process today, a close coupling of discrete and continuous notations in the HyChart style is im-

plemented in the *MaSiEd* tool [AT98], which also offers simulation. [SPP01] explains how the *MaSiEd* notations relate to HyCharts. As a result, the tool can be used as an editor for a HyChart dialect. The *HyTech* tool[8] [HHWT95], which offers model checking of hybrid models, is another element needed as support for an integrated development process. Presently, however, its application is limited due to scalability problems and deficits of the underlying hybrid automata model [MS00]. The new succesor tool *HyperTech* improves the way in which *HyTech* performes arithmetics and which is one reason for the scalability problems [HHMW00]. However, currently there is no performance data available which would cover realistic case studies. Promising tool approaches for the future should couple analysis algorithms such as those implemented in *HyTech/HyperTech* with modular graphical description techniques, e.g., HyCharts, in comprehensive tool frameworks, such as the AutoFocus/QUEST framework for discrete systems.

Integration in evolutionary development processes. While Figure 2.7 at first sight suggests a top-down development process, the integrated development approach is not limited to such processes models. In evolutionary processes [Bal98], the modeling, analysis and refinement steps depicted in Figure 2.7 may be iterated in various design cycles. In this case, the integrated process may be seen as a process pattern within an evolutionary process. In fact, evolutionary development also benefits from the availability of integrated notations and methods for hybrid systems, because they enable an early specification of models which can be evaluated and elaborated in succeeding steps. In contrast, the usage of isolated description techniques similar to Figure 2.5 impedes the evolutionary reuse of models, since formal models can only be specified after a partitioning of a system into discrete-time, discrete-event and continuous-time subsystems. Thus, one is forced to work with rather fine-grained models already in early iteration phases of an evolutionary process. Models given as (informal) text are hardly an alternative here, because they cannot be executed. However, executable prototypes are highly desirable in evolutionary development in order to evaluate a model prior to the next iteration phase.

2.2.3 Supporting Techniques Developed in This Thesis

Description techniques. This thesis is supposed to contribute to the development process outlined above in several ways. It proposes the formal hybrid description technique *HyCharts*, which supports modular specification of hybrid systems (Chapter 3). HyCharts resemble the notation introduced in the software engineering method for real-time object-oriented systems (ROOM) [SGW94], but extend it to the description of hybrid and continuous behavior. ROOM is one basis for the ongoing standardization of the UML [Gro00] dialect for real-time systems, UML-RT [TG00]. As in ROOM and

[8] or other tools, e.g., *Uppaal* or *Kronos* [BLL+96, DOTY96]

in agreeance with the UML's concept of different system views, HyCharts consist of two subnotations: *HyACharts* for the specification of system architecture and *HySCharts* for the specification of component behavior. The main application area for HyCharts ranges from requirements specification to the design phase. Hybrid sequence charts (HySCs), which were outlined above and which complement architecture diagrams in capturing and documenting high-level requirements of hybrid systems, will not be considered further in this thesis. Instead, the methods developed in this thesis focus on the transition from the analysis phase to the design and implementation phases based on HyCharts. For the discrete-time models which are derived from HyCharts and which occur in these later development phases, *DiCharts* are introduced. They exactly correspond to HyCharts but use an underlying discrete-time execution model. In particular, they allow us to closely couple state-transition logic and discrete-time control laws.

Refinement. As far as potential unexpected effects are concerned, a highly critical step in the transition from the analysis phase to design and implmentation is to move from a mixed discrete-event/continuous-time model to a discrete-time model, which enables efficient implementation. The thesis therefore elaborates methods which guide this transition to discrete-time models while maintaining vital classes of properties of the initial model (Chapter 5). An important characteristic of these methods is that they help to make assumptions about the environment explicit. Since they result in constraints on the sampling rate, they can also be used to verify the adequacy of sampling rates present in legacy components or in components developed by third parties w.r.t. a given environment model. Furthermore, we also outline how models can be partitioned into discrete-time and remaining continuous-time and hybrid subsystems, which is useful if a separate development process is intended to be pursued for subsystems that are supposed to be implemented in analog, digital or mixed-signal hardware.

Refinement techniques for architecture and automata diagrams in the style of those in [PR99, Sch98] which do not affect the time model are also considered in the thesis (Chapter 4). However, less emphasis is put on them since [Bro97b] shows that such rules are essentially independent from the underlying time model.

Properties. Obviously, for meaningful refinement techniques it is necessary to examine which classes of properties they maintain. Hence, the thesis studies and classifies typical properties that hybrid systems have to obey (Chapter 6). This provides a deeper insight into the characteristics of hybrid systems and is also used to make relationships to control theory and computer science explicit. As an addition to the general definitions for properties of hybrid sytsems which will be given, the thesis outlines proof methods for some of them and explains why certain classes of properties have not received much attention in computer science so far.

Finally, note that the thesis does not regard validation, refinement and implementation techniques for components only involving state-transition logic or control laws since

techniques for such isolated components do not necessitate hybrid methods.

2.2.4 Other Techniques Supporting Integrated Development

Here, we give a brief overview of further techniques which support the proposed integrated development process. The overview is by no means compresensive, but it lists related work which is particularly relevant for the integrated development process.

Description techniques. In the past, a number of description techniques for hybrid systems have been proposed. As far as formalisms for the definition of hybrid systems' architecture and component behavior are concerned, the reader is refered to Section 3.7.2. Here, we want to concentrate on further formalisms which complement such architecture and behavior descriptions. Concerning the logic-based formalisms, we would like to mention [Lam93, CRH93] and [HMP93]. [Lam93] conservatively extends the temporal logic of actions to hybrid systems by including additional definitions and an axiom which formalizes integration. [CRH93] extends Duration Calculus [CHR92] by admitting propositions over differentials and limit values. To determine the validity of propositions over continuous functions, an appropriate mathematical theory is assumed given. [HMP93] similarly defines an interval temporal logic which contains primitives for derivatives and limits. The authors do not permit formulas which involve function values at isolated time points, but only allow limits and derivatives of functions which are defined over whole intervals. This choice is rational as from an implementation-based point of view, singular values cannot be observed.

[FNW98] defines an extension of UML class diagrams, called UML^h, to hybrid systems. These diagrams distinguish between discrete, continuous, hybrid and abstract classes, which have to be broken down into the other three classes as development progresses. UML^h descriptions can be translated into a hybrid extension of the Z specification language [Fri98b], and, in principle, can also be used in combination with other object-oriented languages for hybrid systems modeling.

The HySC notation mentioned above can be regarded as adopting concepts occuring in timing diagrams [ABHL97], which are widely used in hardware design, and concepts of the *constraint diagrams* introduced in [Die96]. Both these description techniques in turn are based on the idea of visualizing system behavior with trajectories of the state variables.

From the point of view of modeling, the *Ptolemy* project and the associated tool are interesting [Lee01]. There, an environment is being developed which allows the heterogeneous modeling of systems that mix technologies, such as analog and digital electronics, hardware and software, and electronic and mechanical devices. The emphasis is not so much on the coupling of notations, but on the coupling of different computation models. These computation models include clock synchronous continuous-time and discrete-time data-flow, asynchronous message passing and finite state machines.

Ptolemy stems from the electrical engineering domain. For hybrid systems, the coupling of models proposed by Ptolemy is beneficial, because implementation-oriented models of hybrid systems typically exhibit a mixing of technologies.

Analysis and synthesis techniques. Analysis techniques supporting integrated development include model checking tools for hybrid systems like *HyTech/HyperTech* [HHWT95, HHMW00], *Uppaal* [BLL+96] and *Kronos* [DOTY96]. As model checking of hybrid systems is highly sensitive to the complexity of the considered model and in practice often is unfeasable even for small models [MS00], simulation is another important approach to anaysis. An overview over simulation packages for hybrid systems is given in [Mos99].

Lunze et al. try to use stochastic techniques to build a qualitative model of hybrid systems which can be used to derive control recommendations for the operator of a plant [LNR97]. The formal synthesis of discrete controllers for safety monitoring and shutdown of hybrid plants is targeted at in the work of Hanisch et al. [CH99, Han00]. Raisch [Rai98] defines a method for automatically constructing Moore automata as discrete abstractions for (simple) hybrid plants. The granularity of these abstractions, which are guaranteed to contain the behavior of the real system, can be chosen freely. Synthesis techniques for discrete systems can then be applied to construct controllers which supervise the plant [RO98].

Prototyping. A framework for rapid prototyping of (discrete) hard real-time systems is presented in [FKMF97, PMK+99]. It is based on SDL specifications [ITU94] and associated timing constraints. A system can be analysed within the framework by means of prototypes consisting of both hardware and software. When carried over from discrete systems to hybrid systems, such techniques allow the exploration of design alternatives early in the development process, without enforcing detailed design decisions that may be difficult to alter later on.

A related approach, which may in principle be used for the prototyping of HyCharts, is the generation of mixed analog/digital hardware from hybrid data-flow graphs (HDFG) [GW98, Gri99]. HDFG can be used for the (low-level) specification of analog, digital and mixed-signal systems and allow the optimization and generation of hardware from such specifications. For the hardware generation, FPGAs (Field Programmable Gate Arrays) and FPAAs (Field Programmable Analog Arrays) can be selected as a target platform which facilitates the prototyping of mixed-signal systems. A translation from HyCharts to HDFG is developed in [SG99] and [GS00a]. Based on the translation a generation of hardware prototypes from HyCharts becomes feasible.

Design in discrete-time. When discrete-time submodels have been obtained in the integrated development process (Figure 2.7, left branch), synchronous languages such as *Esterel* [Ber96], *Lustre* [HCRP91] and *Signal* [BG90] can be used for further development steps. With respect to this thesis, the main benefit of these languages is the explicit and controllable way in which the progress of time is handled. Basic

paradigms in them are that computation is assumed to be instantaneous unless otherwise specified and that time advances uniformly for all components, i.e. there is an underlying global time scale. These principles are also adopted in HyCharts.

The efficient implementation of discrete-time control laws is a main motivation and focus of the data-flow languages *Lustre* and *Signal*. Therefore, they are ideally suited for this task. For the usage together with the methods and notations developed in this thesis, *Signal* is preferable, because it is also based on a relational modeling paradigm, instead of the functional paradigm of *Lustre* [Cas97]. Apart from that, the clock calculus of *Signal* may be used to efficiently implement discrete-time components which operate at different rates.

Extended state transition diagrams such as *Statecharts* [Har87], *Argos* [Mar91] and *μ-Charts* [Sch98] are alternative candidates for the specification of discrete-time components. However, in the context of HyCharts, DiCharts, which are introduced in Chapter 5 of this thesis, are more appropriate, since their computation model is simpler and matches directly with HyCharts. Furthermore, they allow the specification of discrete-time control laws which are specific to a control state. For the efficient implementation of DiCharts, the imperative parallel programming language *Esterel* is well suited. A translation from a DiCharts variant to an extension of *Esterel* has been presented in [SSH99]. It also makes DiCharts amenable to the verification methods implemented in the system *C@S* [SK97].

Systems engineering. Parnas et al. suggest a systematic way to use functional documents in the systems engineering process [PM95]. The authors define the information which should be contained in documents like the *system requirements document*, describe various kinds of documents relating to different system entities at different levels of granularity, and reveal the relationship between the information in these documents. Thus, this work can serve as a general reference on how notations and development techniques, like the ones developed in this thesis, can be used in practical systems development. Note that the authors also emphasize the role of documenting not only the interface but also the behavior of a system's environment, and mention that their requirements model corresponds to control theory. The notations and methods introduced in this thesis can be regarded as a class of the domain specific techniques which are proposed in [PM95]. Similar ideas which are even closer to this thesis can be found in [Bro97c]. The Four Variable Model and SCR, which form the background of [PM95], are discussed in the specific context of hybrid systems in [EKM+93] and [Hei96].

2.3 Discussion and Further Work

We have outlined elements of an integrated development process for hybrid systems that is based on precise notations and formal methods. In contrast to a conventional

process, it aims at reducing design risks by earlier validation of abstract, integrated system models, before implementation-oriented decisions must be made. Moreover, it enables more precise documentation. The use of integrated notations in this process furthermore is a prerequisite for obtaining greater design automation for hybrid systems, since automated techniques can only regard the discrete and the continuous aspects of hybrid systems if their input contains both these aspects.

Based on the description of the analysis and design phases in the integrated process, the chapter explained how this thesis contributes to the applicability of an integrated process and how related work contributes to it.

Note that from the point of view of today's engineers, the conventional develoment process remains reasonable as long as there are no tools which are suitable for industrial practice and which support an integrated process. One has to bear in mind, though, that with increasing system complexity, design risks in the conventional process also increase.

2.3.1 Related Work

The aspect of postponing the partitioning of a system into discrete-time and continuous-time parts is related to the area of hardware/software codesign [BR95]. There, the decision on which parts of a system are implemented in hardware and software is postponed to later phases. However, unlike hardware/software codesign, the partitioning into discrete-time and continuous-time components proposed in the integrated development process does not imply whether the components are implemented in hardware or software. The discrete-time part can be implemented in software or on digital hardware. Even the continuous-time part could still be turned into a discrete-time model and implemented in software (or digital hardware), or it can be implemented in analog hardware. Thus, hardware/software codesign is largely complementary to the proposed integrated development process. Hardware/sofware codesign predominantly aims at the design of digital systems and usually assumes a system specification to be given. Such a specification of digital (or rather discrete-time) system components is a result of the integrated process. Nevertheless, the integrated process can be regarded as involving an *analog/digital codesign* process, because it first involves notations which do not yet imply analog or digital implementation and then employs (refinement) techniques to obtain such implementations in later design phases.

The development process for hybrid systems proposed in [CWM98] can be regarded as an intermediary between the two processes outlined here. The authors propose complementing block diagrams and automata-based notations with formal specifications using Z [Spi92].

Sinclair et al. [Sin97, HS97] propose a development process similar to the integrated process presented here. The process basically carries over object-oriented modeling to hybrid systems. It relies on UML-style notations in the analysis phase and uses

hybrid automata for detailed design. Unfortunately, the work remains rather superficial. A partitioning of models according to different time models and according to state-transition and control aspects is not addressed. In our view, hybrid automata are not as close to implementation as the authors suggest. Namely, their precise implementation in software is impossible as long as the taking of transition is not bound to a discrete time grid. We will adress this problem for HyCharts in this thesis. Furthermore, [Sin97] and [HS97] do not explicitly discuss benefits of the integrated process w.r.t. today's practice, and the possible integration of existing techniques for the state-transition part and the control part remains open. For the practical feasibility of an integrated process, an interface to existing conventional techniques is vital in our view.

Similar to our integrated development process, Hung et. al [HG96] also propose moving from abstract specifications in continuous-time to discrete-time implementations. Based on Duration Calculus [CHR92], the authors introduce some rules supporting this methodology. The rules allow the deduction of the validity of duration calculus formulas in discrete-time from the validity of the formulas in continuous-time, if the boolean variables in the formulas satisfy a certain restriction on the frequency with which they may change. However, this paper does not consider continuous dynamics, but instead focuses on the real time aspects.

2.3.2 Further Work

As far as the methodology for hybrid systems development is concerned, we think that the closer integration of sequence chart notations is promising as a link to informal, textual requirements. Thus, this integration should be explored further. Besides this, there still is a lack of methods which can be used to guarantee properties of hybrid systems that affect their state-transition logic as well as control laws.

Chapter 3

HyCharts – Specification of Hybrid Behavior

In this chapter we introduce *HyCharts* as specification techniques for hybrid systems, and define their semantic foundations. Over the past few years a number of specification techniques have been developed for hybrid systems. While they are all well suited when a software system and its environment are modeled as one unit, the search for hybrid description techniques which allow specification based on individual components is relatively new.

For such a component-based view modularity is essential. It is not only a means for decomposing a specification into manageable small parts, but also a prerequisite for reasoning about the parts individually, without having to consider the interior of other parts. Thus, it greatly facilitates the design process and can help to push the limits of verification tools, like model-checkers, further.

With a simple and powerful computation model for hybrid systems and with a collection of operators on hierarchic graphs as tool-set, we follow the ideas in [GSB98b] and define HyCharts. They consist of two different relational interpretations of hierarchic graphs, an *additive* one and a *multiplicative* one. Under the additive interpretation the graphs are called *HySCharts* and under the multiplicative one they are called *HyACharts*. HySCharts are a visual representation of hybrid, hierarchic *state transition* diagrams. They model the control-flow within hybrid components. HyACharts are a visual representation of hybrid *data-flow* graphs (or architecture graphs). They model the data-flow between the components of a hybrid system and allow the designer to compose components in a modular way. The behavior of these components can be described by using HySCharts or by any technique from system theory that can be given a compatible semantics, i.e. a semantics in terms of *dense input/output relations* (introduced below). This includes differential equations. Simple syntactic transformations, corresponding to macro expansion, lead from the notation used by the designer to a hierarchic graph whose semantics results from the respective interpretation of the

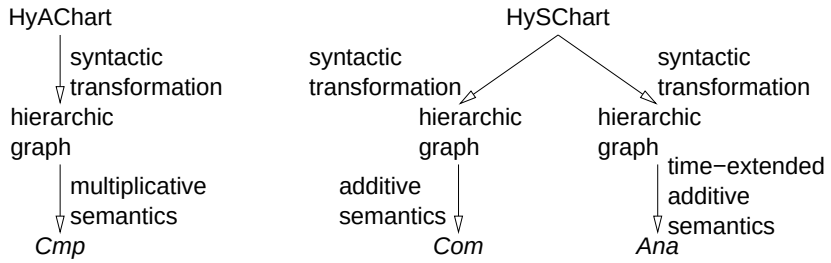


Figure 3.1: Principle of the semantics definition for HyCharts.

graph. For HyACharts, the syntactic transformation is trivial and applying the multiplicative interpretation to the resulting graph yields the semantics of a component *Cmp* (Figure 3.1, left). For HySCharts, transformation is more complex. Here, two transformations are necessary, one to extract the discrete dynamics from the diagram and one to extract the analog dynamics (Figure 3.1, right). For the analog dynamics, a time-extended variant of the additive interpretation yields its semantics *Ana*. For the discrete dynamics, the additive interpretation directly yields its semantics *Com*. The coupling of *Ana* and *Com* is formalized in our hybrid computation model.

The algebra-based semantics which maps graphs to relations has three main advantages. First, up to (rather simple) syntactic transformations, it corresponds almost one-to-one with the visual notation used by software engineers. Second, as shown in [GBSS98, GBSS99], it comes equipped with a set of graph equations (algebra) which define how to (visually) transform components in a semantics preserving way. As a result, the algebra may be used by engineers both for optimizations and to check the equivalence of different components. Third, similarly to [AH97, Bro97b], it comes equipped with a very simple notion of refinement and its associated compositional refinement rules. This is an essential prerequisite for proving that a successively modified implementation meets its original specification. We will consider specific refinement techniques for components in Chapters 4 and 5 of the thesis. In this chapter, we present the infrastructure of our visual notation and its textual representation. Together with refinement techniques, it allows the hierarchic specification and analysis of hybrid systems.

Publication history. The core of this work has been published in [GS00b]. The presentation we offer here differs in some details, however. First, we integrate invariants into HySCharts, which results in a more liberal semantics for transitions that is needed in Chapter 5. This adaptation was published in [Sta00a]. Second, the multiplicative interpretation of hierarchic graphs is defined for a continuous *and* for a discrete model of time here, and the time-extended additive interpretation is introduced more consequently and also for different time models. The different time models are necessary for Chapter 5 and for some proofs in Appendix A. Third, in this presentation we explicitly allow feedback without delay in architecture diagrams, as long as the feedback is well

defined. This is reasonable, as control engineers are familiar with the problems arising from instantaneous feedback (or *algebraic loops*). Fourth, continuous activities are formalized in greater detail. This is needed for a formalization of inductive reasoning for HySCharts which is given in Appendix A.1.2. Implicitly, this inductive principle has already been used in earlier papers, but without explicit formal foundation. Finally, we introduce a finer classification of communication channels and variable classes here. The classification is a basis for the refinement method developed in Chapter 5.

Overview. This chapter is organized as follows. Section 3.1 provides an initial impression of HyCharts by means of an example. The hybrid computation model, which provides some insight into the dynamics of hybrid systems and which is the basis for the semantics of HySCharts, is explained in Section 3.2. In Section 3.3, we present two abstract, basic interpretations of hierarchic graphs and a third, derived interpretation. These interpretations provide the infrastructure for defining a surprisingly simple denotational semantics for the key concepts of Statecharts [Har87] offered in HyCharts, e.g. hierarchy and preemption. The derived interpretation covers the switched continuous dynamics expressible with HyCharts. Following the ideas developed in the hybrid computation model, HyCharts are defined in Sections 3.4 and 3.5 as a multiplicative (data-flow) and an additive (control-flow) interpretation of hierarchic graphs, respectively. Both diagram types are introduced by way of using the example of Section 3.1. In Section 3.6, we briefly discuss how other techniques for component specification can be integrated into our approach. Finally, in Section 3.7, we summarize our results and relate them to the literature.

Note that some parts of this presentation require a knowledge of mathematical topology. In case of concepts not known to the reader, he or she is is referred to Appendix B.2 where all relevant concepts can be looked up. For most parts, an intuitive understanding of the natural topology on the real numbers suffices.

3.1 An Example

The following example illustrates the kinds of systems we target at. It will be used throughout the thesis to demonstrate the use of the notations and methods developed here. Note that we do not explain any details of the HyChart specification for the example system in this section. Instead the presentation here aims at providing a first look and feel of HyCharts.

Example 3.1 (Electronic height control system, EHC.) The purpose of the electronic height control system (EHC), which was originally proposed by BMW, is to control the chassis level of an automobile by a pneumatic suspension. The abstract model of this system, which considers only one wheel, was first presented in [SMF97]. It basically works as follows:

Whenever the chassis level is below a certain lower bound, a *compressor* is used to increase it. If the level is too high, air is blown off by opening an *escape valve*. The chassis level *sHeight* is measured by *sensors* and *filtered* to eliminate noise. The filtered value *fHeight* is read periodically by the *controller* which operates the compressor and the escape valve and resets the filter when necessary. A further sensor, *bend*, tells the controller whether the car is going through a curve.

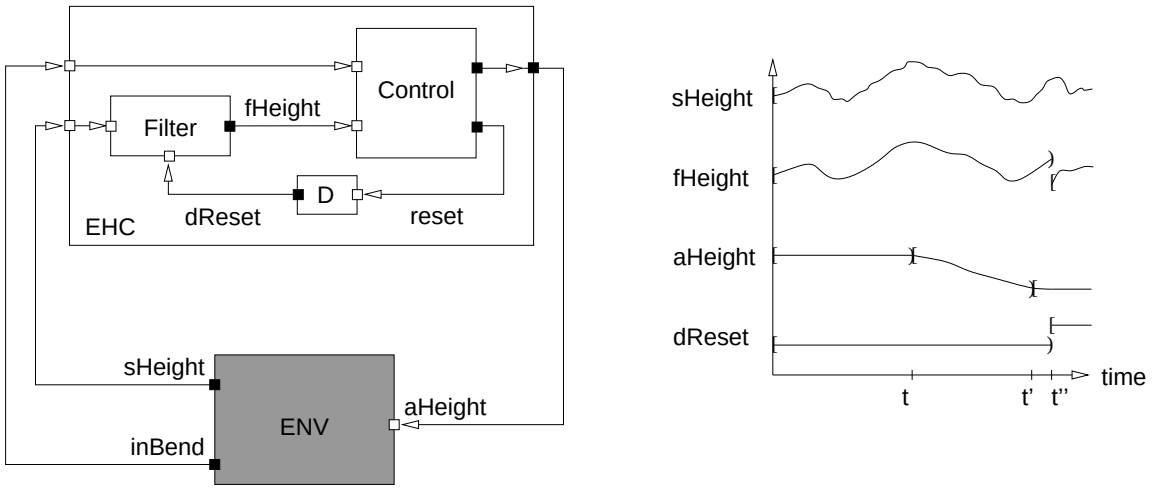


Figure 3.2: The EHC: Architecture and a typical evolution.

The diagram in Figure 3.2, left, depicts the architecture of the EHC and its interconnection to the environment. The environment, shaded in grey in the figure, will not be regarded in much detail in the thesis. Realistic models for disturbances caused by driving on a road can be found in [KL94]. In this chapter we concentrate on the open system consisting of the filter, the controller and a delay element that ensures that the feedback is well defined. The escape valve and the compressor are not modeled explicitly here, but are implicitly contained in the controller.

Data-flow diagrams like the one in Figure 3.2, left, are called HyACharts. Each component of such a chart can be defined again by a HyAChart or by a HySChart or some other compatible formalism. The components only interact via clearly defined interfaces, namely channels, which results in a modular specification technique.

The behavior of a component is characterized, as intuitively shown in Figure 3.2, right, by periods where the values on the channels change smoothly and by time instances in which there are discontinuities. In our approach, the smooth periods result from the analog parts of the components. The discontinuities are caused by their discrete parts. Sometimes we also call these discrete parts *combinational parts* to emphasize that they contain no memory, but can be regarded as combinational circuits.

We specify the behavior of both the discrete and the analog part of a component within a single HySChart, i.e. by a hybrid, hierarchic state transition diagram, with

states marked by activities and transitions marked by actions. The transitions define the discontinuities, i.e. the instantaneous actions performed by the discrete part. The activities define the smooth periods, i.e. the time consuming behavior of the analog part while the discrete part is idle. The discrete part is idle between the time instances at which it performs actions. As an example, Figure 3.3 shows the HySChart for the

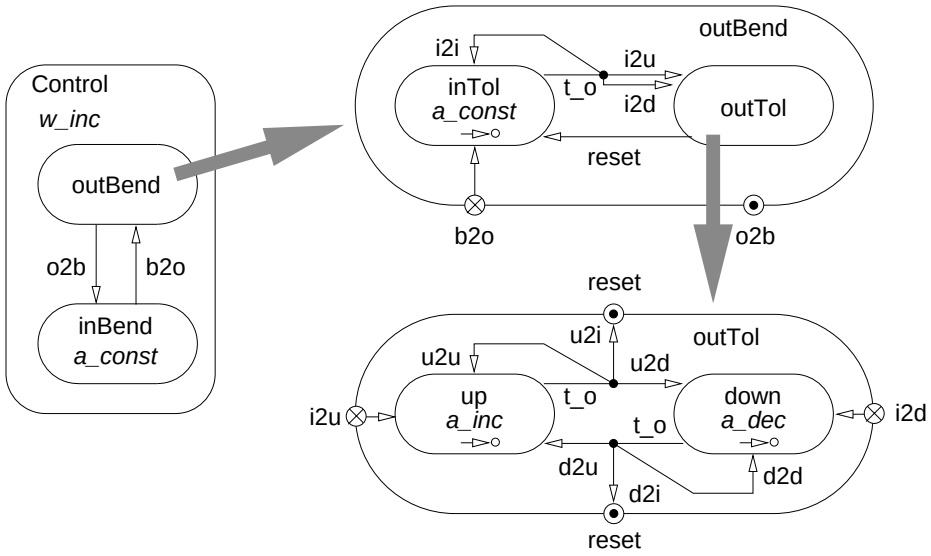


Figure 3.3: The EHC's *Control* component.

EHC's *Control* component. The nodes in the figure represent (control) states and the arcs represent transitions. In the following we will informally explain the control-flow in the diagram. The details are elaborated in Section 3.5. The HySChart in Figure 3.3 consists of three hierarchic levels.[1] Figure 3.3, left, depicts the highest hierarchic level, with the substates *outBend* and *inBend*. Control is in *inBend* when the car is going through a curve, otherwise it is in *outBend*. This hierarchic state is refined into the substates *outTol* and *inTol*, depicted in Figure 3.3, top right. Control resides in *inTol* as long as the chassis level is within a given tolerance interval. If the chassis level is outside the interval, one of the two substates of *outTol*, *up* or *down*, is entered. Figure 3.3, bottom right, shows this refinement of *outBend*. The labels at the transitions are called actions and refer to predicates which define when the transition can be taken and how it affects the HySChart's data state. For example, the action $i2i$ expresses that the chassis level must be inside the tolerance interval when the transition is taken and that the transition leaves the component's variables unchanged. The activities, written in italics in the figures, refer to predicates which describe the continuous evolution of the component's variables while control is in the respective state. Activity *a_const* of states *inBend* and *inTol*, for instance, refers

[1] In the figure we use large grey arrows, which are *not* part of the HyChart notation, to indicate the connection between the hierarchic levels.

to a predicate specifying that variable *aHeight* remains constant. In the model, this expresses that compressor and escape valve are turned of. Hence, the EHC system does not actively modify the chassis level in states with this activity. Besides the activities, we furthermore associate a state invariant with every state in the diagram. The predicate defining such an invariant identifies all those variable evaluations for which control may reside in the respective state. Basically the invariant can be derived from the outgoing transitions of a state. The invariant of *inTol*, for instance, expresses that the chassis level is in the given tolerance interval. We will explain the actions, activities and invariants in more detail in Section 3.5. □

3.2 The Hybrid Computation Model

We start this section by explaining informally how our hybrid computation model works. After that, the model's constituents are introduced formally.

3.2.1 General Idea

We model a *hybrid system* by a network of autonomous *components* that communicate via one-directional channels in a *time synchronous* way (see Figure 3.4, left, where boxes denote components). Time synchrony is achieved by letting time flow uniformly for all components. It facilitates reasoning about time, since no local drifting clocks have to be considered.[2] Furthermore, time synchrony suits well to the block diagrams from control theory (Section 2.1.2).

Component network. In a hybrid system the data-flow between components may be continuous (think of analog devices), so we assume that time increases continuously and take the non-negative real numbers $\mathbb{R}_+$ as the abstract time axis. The data exchanged along a channel with type A over time defines a mapping $a \in \mathbb{R}_+ \to A$, with certain continuity restrictions which we will elaborate in Section 3.2.3. We call such a restricted mapping a *dense communication history* (or *dense stream*). On the level of semantics the behavior of a component can be completely described by an *input/output relation*, i.e. by a relation between the histories of its input channels and the histories of its output channels. Relations are used instead of functions, because this allows us to express nondeterminism. The relations must be total in the input histories, i.e. for every input history an output history must exist which is related to the input history by the relation. Due to this, we often use the functional notation $c(a)$ to denote the set of all elements that is related to a by c, formally $c(a) = \{b \mid (a, b) \in c\}$. Furthermore, we assume that the relations are defined such that the data occurring

[2] Nevertheless HyCharts allow us to explicitly model drifting clocks [Sta99].

in the histories of the output channels up to time t only depends on the input history received up to t.[3] Formally, for component semantics c, all a_1, a_2 and t:

$$a_1|_{[0,t)} = a_2|_{[0,t)} \quad \Rightarrow \quad c(a_1)|_{[0,t)} = c(a_2)|_{[0,t)}$$

where by $a|_\delta$ we denote the restriction of a to the time interval $\delta \subseteq \mathbb{R}_+$. Restriction is overloaded to sets of streams, like $c(a)$, in a pointwise manner. We call these relations *time guarded.* Informally, time guardedness expresses that a component does not anticipate future input. Clearly, every realizable component, i.e. every component which can be implemented, behaves in this way.

Machine model. Each component is modeled by a *hybrid machine*, as shown in Figure 3.4, middle. This machine consists of five parts: a *discrete* (or combinational) part (*Com*), an *analog* (or continuous) part (*Ana*), a *feedback* loop, an infinitesimal delay (Lim_s), and a projection (*Out*). The feedback and Lim_s model the *state* of the

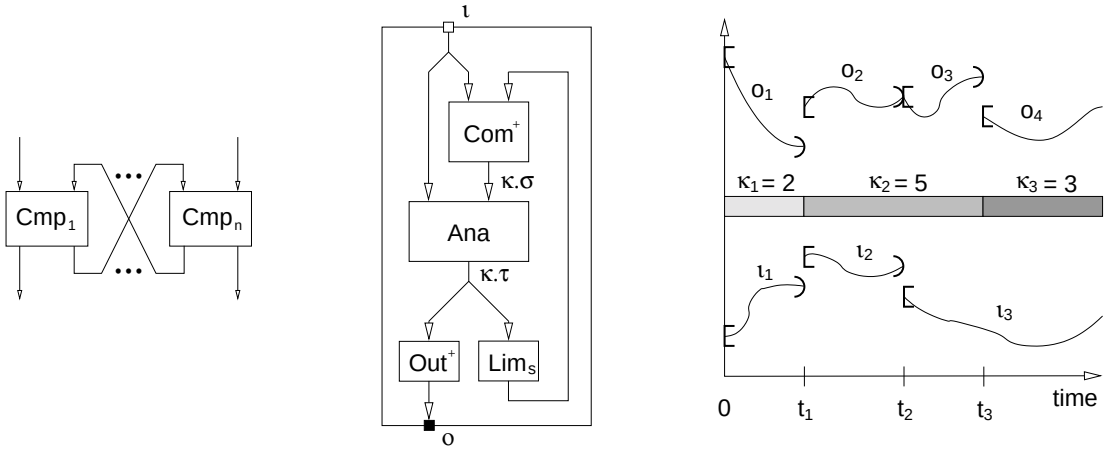


Figure 3.4: Component network and hybrid machine computation model.

machine. They allow the component to remember at each moment in time t the data computed "just before" t.

The discrete part contains the state-transition logic. It is concerned with the control of the analog part and has *no memory*. It instantaneously and nondeterministically maps the current input and the fed back state to the next state. The next state is used by the analog part to select an *activity* among a set of activities (or execution modes) and it is the starting state for this activity. If the discrete part passes the fed back state without modification, we say that it is *idle.* The discrete part can only select a new next state (different from the fed back state) at distinct points in time. During the intervals between these time instances it is idle and the selection of the corresponding activity is stable for that interval.

[3]Note that in control theory the stronger requirement of demanding that the output at time t already is determined by the input received up to (but *excluding*) t is often used, see e.g. [Son90].

The analog part describes the input/output behavior of the component whenever the discrete part is idle. Hence, it adds the temporal dimension to the component. It may select a new activity whenever there is a discontinuity in the input it receives from the discrete part. An activity can be seen as a set of trajectories which describe the evolution of the components' variables in dependence on the external input.

Example 3.2 Figure 3.4, right, shows the exemplary behavior of a component. The grey boxes labeled with κ_i indicate the time periods where the discrete part idles in control state κ_i. At time t_1 the discrete move of the environment triggers a discrete move of the discrete part. According to the new next state received from the discrete part, the analog part selects a new activity. The activity's start value at time t_1 is as determined by the discrete part. At time t_2 there is a discrete move of the environment, but the discrete part remains idle. Therefore the current activity is not changed, but the analog part chooses a new trajectory from it. The start value of this trajectory is the analog part's output just before t_2, because this is what it receives from the discrete part at time t_2 (due to idling of the discerte part). Thus, the output has a higher order discontinuity here. At time t_3 the environment does not perform a discrete move, but the discrete part does, e.g., because some threshold is exceeded. Again the analog part selects a new activity, which begins with the start value determined by the discrete part. During the intervals $(0, t_1), (t_1, t_3)$ and (t_3, ∞) the discrete part is idle. □

Feedback and state. Since the input received and the output produced may change abruptly at any time t, as shown in Figure 3.4, right, we consider that the state of the component at moment t is the limit from the left $\lim_{x \nearrow t} \psi(x)$ of all the outputs $\psi(x)$ produced by the analog part when x approaches t.[4] In other words, the feedback loop reproduces the analog part's output with an infinitesimal *inertia*. We say that the output is *latched*. The infinitesimal *inertia* is realized by the Lim_s part of the hybrid machine (Fig. 3.4, middle). Its definition is:

$$Lim_s(\psi)(t) \quad = \quad \begin{cases} s & \text{if } t = 0 \\ \lim_{x \nearrow t} \psi(x) & \text{if } t > 0 \end{cases}$$

where s is the initial state of the hybrid machine.

The *data state* of the machine consists of a mapping of *latched* (or *controlled*) variable names to values of corresponding type. Let S denote the set of controlled variable names with associated domains $\{\sigma_v \mid v \in S\}$. Then the set of all possible data states, the *data-state space*, is given by $\mathcal{S} = \prod_{v \in S} \sigma_v$.

The set of controlled variable names can be split in two disjoint sets: a set P of *private* variable names and a set O of *output* (or *interface*) variable names. We write $\mathcal{L}$ for $\prod_{v \in P} \sigma_v$ and $\mathcal{O}$ for $\prod_{v \in O} \sigma_v$. Clearly, $\mathcal{S} = \mathcal{L} \times \mathcal{O}$. $\mathcal{O}$ is also called the *output*

[4] The continuity restrictions we enforce in the following section ensure that the limit from the left is always well defined.

space. Some external inputs, namely the time stamps which are associated with certain kinds of input channels, are also copied into private variables in P. We also call these variables the *latched time stamps*. As we will see this allows the detection of events and discrete jumps. The controlled variables furthermore include a special variable *now* which contains at each moment the current time. Figure 3.5 depicts this classification of the controlled variables.

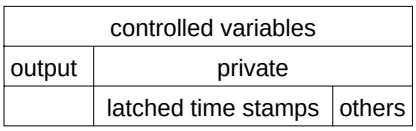


Figure 3.5: Classification of the controlled variables.

The input is a mapping of *input* variable names to values of corresponding type. Let I denote the set of input variable names with associated domains $\{\sigma_v \mid v \in I\}$. Then the set of all possible inputs, the *input space*, is defined by $\mathcal{I} = \prod_{v \in I} \sigma_v$.

3.2.2 The Discrete Part

The discrete part is a relation from the current inputs and the latched state to the next state, formally:

$$Com \in (\mathcal{I} \times n \cdot \mathcal{S}) \to \mathcal{P}(n \cdot \mathcal{S})$$

where $\mathcal{P}(X) = \{Y \subseteq X \mid Y \neq \{\}\}$ and $n \cdot \mathcal{S}$ is the *program-state space*. The program-state space is introduced in technical detail in Section 3.3.3.1. For the time being, a program state may be regarded as consisting of a *control state* $k \in \{1, \ldots, n\}$ and a data state $s \in \mathcal{S}$. As we will see n is the number of leaf nodes in the hierarchic graph that defines *Com* (see Section 3.5.1). The discrete part corresponds to the state-transition relation in ordinary automata models. Its computation takes no time.

An important property of the relation defining the discrete part is that it is defined for all states and inputs, i.e. it is *total*. To emphasize totality, we wrote it in a functional style. Nevertheless, note that for any input and latched state there may be more than one possible next state. This allows us to express nondeterminism. When the output of *Com* for given input i and state s contains the original state s, i.e. $s \in Com(i, s)$, we say that it *can idle* for i and s. If for present state s and input i, next state $s \in Com(i, s)$ is indeed selected in the execution thread under consideration, we say that *Com idles* or *is idle* at that point in time. Finally, we require that the set $CI \subseteq \mathcal{I} \times n \cdot \mathcal{S}$ of inputs and states for which *Com* can idle be topologically open.[5]

[5] As topology we use the Tychonoff topology on $\mathcal{I} \times n \cdot \mathcal{S}$ which is induced by (1) using the discrete topologies on the variable domains different from $\mathbb{R}$ and on the domains of variables that denote time stamps, and (2) by using the natural topology on the real line for domain $\mathbb{R}$ of the remaining variables (see Appendix B.2).

This guarantees that if *Com* idles it can remain idle for some time $t > 0$. Informally, arguing on the real numbers, the reason for this is that any point in an open set has a neighborhood which also is in the set. This property is needed in Section 3.2.4 to ensure that the semantics of a hybrid machine is well defined.

Note again that there is no memory in the discrete part. The latched state is exclusively stored in the *Lim* component of the machine model and given to *Com* after an infinitesimal delay. In the machine model, the discrete part is active at any point in time (this is formalized in Section 3.2.4). It can produce a new next state without being bound to any kind of time grid.

3.2.3 The Analog Part

Whenever the discrete part idles, the analog part performs an *activity*. We describe an activity by a relation *Act* with type:

$$Act \quad \subseteq \quad (\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}}) \times \mathcal{S}^{\mathbb{R}_{s+}}$$

For any set M, the set $M^{\mathbb{R}_{s+}}$ stands for the set of functions from the non-negative real numbers $\mathbb{R}_+$ to M that are *Lipschitz continuous* and *smooth.* We say that a function $f \in \mathbb{R}_+ \to M$ is smooth iff f is infinitely differentiable (i.e., f is in C^∞) for $M = \mathbb{R}$ or f is constant for $M \neq \mathbb{R}$. Infinite differentiability is required for convenience. It allows us to assume that all differentials of f are well defined. Lipschitz continuity provides that the limit from the left of these functions exists. For $M = \mathbb{R}$ Lipschitz continuity is defined as usual in mathematics (Appendix B.3, Definition B.15). For $M \neq \mathbb{R}$ a function $f \in \mathbb{R}_+ \to M$ is Lipschitz continuous iff it is constant. This corresponds to regarding M as a *discrete metric space* (see Appendix B.3, Definition B.12). A tuple of functions is Lipschitz continuous or smooth, respectively, iff all its components are. We also call $M^{\mathbb{R}_{s+}}$ the set of *flows* over M. For $((\iota, \varphi), \psi) \in Act$ the components ι and φ are regarded as input, and ψ is the output of *Act*. Stream ι is also called the (external) input stream, φ the received data-state stream and ψ the output data-state stream. We write the type of activities in a relational style to emphasize that they are not total in their input. Instead a different condition is required, as we will see below. Nevertheless, we will also use the functional notation $\psi \in Act(\iota, \varphi)$ in the following if *Act* is nonempty for ι and φ. Note that we will also use this convention for other relations.

To model analog behavior in a "well behaved" way, activities must be time guarded. As for components, this is required in order to prohibit that activities anticipate future input. It also is a common assumption in control theory where is precludes control laws which cannot be implemented. Furthermore, we demand that the activities do not depend on absolute time (measured from system start) but may be started anytime. This formally means that for all time intervals $[u, v)$, $u, v \in \mathbb{R}_+$, and for all histories $\varphi, \psi \in \mathcal{S}^{\mathbb{R}_{s+}}$ and $\iota \in \mathcal{I}^{\mathbb{R}_{s+}}$:

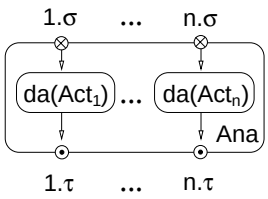


Figure 3.6: Composition of activities.

$$((\iota,\varphi),\psi)|_{[u,v)} \in Act|_{[u,v)} \;\Rightarrow\; \forall t \geq -u.\ ((\iota^t,\varphi^t),\psi^t)|_{[u+t,v+t)} \in Act|_{[u+t,v+t)}$$

where φ^t is the right shift of stream φ by t, $\varphi^t(x) = \varphi(x-t)$ and the time restriction $R|_\delta$ of a relation (or set) R is defined in a pointwise manner, i.e. $R|_\delta = \{r|_\delta \mid r \in R\}$. Independence from absolute time also is a common assumption in control theory. It simplifies the analysis of activities, because absolute time need not be considered. Any point $((\iota,\varphi),\psi)(t)$ visited by a tuple of histories $((\iota,\varphi),\psi)$ in Act can be taken as new start value.

The complete behavior of the analog part is described by a relation Ana with type:

$$Ana \quad \subseteq \quad (\mathcal{I}^{\mathbb{R}_{p+}} \times (n \cdot \mathcal{S})^{\mathbb{R}_{p+}}) \times (n \cdot \mathcal{S})^{\mathbb{R}_{p+}}$$

where $n \cdot \mathcal{S}$ is the program-state space, as in the type of Com, and for any set M, $M^{\mathbb{R}_{p+}}$ denotes the set of *piecewise smooth, piecewise Lipschitz continuous* functions in $\mathbb{R}_+ \to M$. We call a function $f \in \mathbb{R}_+ \to M$ piecewise smooth and piecewise Lipschitz continuous iff every finite interval on the non-negative real line $\mathbb{R}_+$ can be partitioned into *finitely* many left closed and right open subintervals such that f is smooth and Lipschitz continuous on each subinterval. Like activities, Ana is not total and for $((\iota,\sigma),\tau) \in Ana$ we again regard ι and σ as input and τ as output. Due to its type the input and output of the analog part is not necessarily continuous. Instead, finitely many discrete moves by the discrete part and the environment during any finite interval are allowed. In the following we will see that this demands that the discrete part is realizable. We call $M^{\mathbb{R}_{p+}}$ the set of *dense communication histories* or *dense streams*.

The relation Ana is obtained by pasting together the flows of the activities associated with the control states where the discrete part Com idles. Pasting is realized by first adapting activities such that they permit discontinuities in their output date-state stream whenever there are (higher-order) discontinuities in the input stream or data-state stream they receive. The adapted activities are then composed with a switching operator, called the *time-extended disjoint sum*, as indicated in Figure 3.6.[6] For activity Act, its adaptation to discontinuities $da(Act)$ is defined as follows:

$$\begin{aligned} da(Act) \quad = \quad \{\ & ((\iota,\sigma),\tau) \in (\mathcal{I}^{\mathbb{R}_{p+}} \times \mathcal{S}^{\mathbb{R}_{p+}}) \times \mathcal{S}^{\mathbb{R}_{p+}} \mid \\ & \forall \delta \in Int.\ (\iota,\sigma)|_\delta \in (\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}})|_\delta \Rightarrow ((\iota,\sigma),\tau)|_\delta \in Act|_\delta\} \end{aligned}$$

[6] We use boxes with rounded corners for nodes in hierarchic graphs under the additive or time-extended additive interpretation (see Sections 3.3.3 and 3.3.4).

where *Int* denotes the set of all left closed and right open intervals over $\mathbb{R}_+$. Informally, the formula expresses that whenever ι and σ are smooth and Lipschitz continuous (left side of the implication) then τ also is. If ι or σ restricted to one particular interval is discontinuous, no restriction of τ results for this interval. In other words, the universal quantification over intervals leads to restrictions on τ exactly on those intervals where ι and σ are smooth and Lipschitz continuous. Therefore, τ may be discontinuous whereever ι or σ is. The type of $da(Act)$ is $(\mathcal{I}^{\mathbb{R}_{p+}} \times \mathcal{S}^{\mathbb{R}_{p+}}) \times \mathcal{S}^{\mathbb{R}_{p+}}$. The time-extended disjoint sum with which the adapted activities are composed to yield the analog part will be defined in Section 3.3.4. Here we only explain its principle. For $((\iota, \sigma), \tau) \in Ana$ the sum uses to control information contained in the stream of program states σ to demultiplex the data state information to the adapted activity which is associated with the given control state. This activity determines the data state information in τ. The sum defines the control information in τ by setting it equal to the control information in σ. This can be seen as multiplexing the output of the selected activity by adding the control information. Like the discontinuity adaptation, the time-extended disjoint sum operation is also defined by regarding its input and output over time intervals, not just time points.

As we demand that every activity is time guarded and as discontinuity adaptation and the disjoint sum operation also do not anticipate future input, the analog part is time guarded. Furthermore, for the analog part we demand that it is *resolvable*, which means that it must have a fixed point for every start state $s_0 \in n \cdot \mathcal{S}$ and every input stream $\iota \in \mathcal{I}^{\mathbb{R}_{s+}}$, i.e.:

$$\exists \sigma \in (n \cdot \mathcal{S})^{\mathbb{R}_{s+}} . \sigma(0) = s_0 \ \wedge \ \sigma \in Ana(\iota, \sigma)$$

Resolvability of the analog part is needed to prove that the semantics of a hybrid machine is well defined (see below). Informally, it is needed, because when the discrete part of the machine model idles, the machine model is equivalent to a direct feedback over *Ana*. Resolvability is required instead of totality. In Section 3.5.2, we explain conditions under which resolvability of individual activities results in a resolvable analog part.[7]

3.2.4 Semantics of the Hybrid Computation Model

Given an initial state s_0, the behavior of the hybrid machine is a relation *Cmp* between its input and output communication histories. The denotational semantics of *Cmp* directly results from writing the graph in Figure 3.4, middle, as a relational expression with the operators which will be defined in Section 3.3.2. For the purpose of this section we expand the definitions of these operators and introduce a further relation *St* denoting the "white-box" (or state-based) semantics of a component, without the output projection *Out*. The resulting definition for *Cmp* and *St* is:

[7] Resolvability of activities is similar to the above definition of resolvability for the analog part.

$$
\begin{array}{lcl}
St & \in & n \cdot \mathcal{S} \to (\mathcal{I}^{\mathbb{R}_{p+}} \to \mathcal{P}((n \cdot \mathcal{S})^{\mathbb{R}_{p+}})) \\
St(s)(\iota) & = & \{\tau \in (n \cdot \mathcal{S})^{\mathbb{R}_{p+}} \mid \exists \sigma \in (n \cdot \mathcal{S})^{\mathbb{R}_{p+}}. \\
 & & \qquad \sigma \in Com^{\dagger}(\iota, Lim_s(\tau)) \ \wedge \\
 & & \qquad \tau \in Ana(\iota, \sigma)\} \\
Cmp & \in & n \cdot \mathcal{S} \to (\mathcal{I}^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_{p+}})) \\
Cmp(s)(\iota) & = & Out^{\dagger}(St(s)(\iota))
\end{array}
$$

where $R^{\dagger}$ trivially extends the relation $R \in A \to \mathcal{P}(B)$ to streams, i.e. $R^{\dagger}(\alpha) = \{\beta \mid \forall t \in \mathbb{R}_{+}.\beta(t) \in R(\alpha(t))\}$. *Out* is a projection which selects the output variables from the state. It is extended to sets of states in a pointwise manner and extended in time by †. Furthermore, note that Lim_s is deterministic. Hence, $Lim_s(\tau)$ is a single stream.

In Appendix A.1.2 we also define a semantics for components that is based on transition systems and prove its equivalence to the denotational semantics (Theorem A.5). The transition system semantics formalizes our informal arguing over the behavior of a component as caused by discrete moves of *Com* and periods of continuous evolution due to *Ana*. It also is the basis for inductive reasoning over components (Theorem A.6). Moreover, Appendix A.1.3 proves that *Cmp* is a *time-guarded* relation.

Due to its importance and in order to foster a deeper understanding of the machine model we now show that *Cmp* is total. This amounts to proving the existence of a fixed point in the above definition for arbitrary starting state and input. As the definition involves feedback between *Com* and *Ana* which are coupled without a finitely large delay $\delta > 0$ and as *Ana* is not total, the existence of a fixed point is not guaranteed a priori. Instead, it is a consequence of the properties of *Com* and *Ana*. Note that for a better understanding of the machine model, the reader unfamiliar with topology need not follow the topological details, but may instead focus on the structure of the proof.

Proof. (Existence of a fixed point.) First, we prove that some time $t > 0$ passes between two discrete moves by the discrete part or the environment, i.e. between two points in times where *Com* is not idle or the input has a discontinuity. Suppose s' is an output of *Com* for the current input i and the latched state s at time t_0 (see Figure 3.7). Then, *Com* can idle for the new state s', i.e. $s' \in Com(i, s')$. This

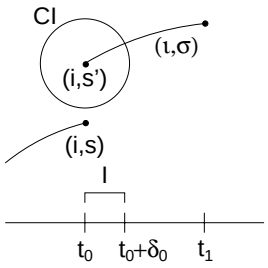


Figure 3.7: Computing the possible delay.

holds due to the construction of *Com* from the HySChart and will be justified in

Section 3.5.1.6. *Com* can remain idle as long as its input from the environment and the feedback loop are still in *CI*. (As introduced in Section 3.2.2, *CI* is the set of inputs and states for which *Com* can idle.) Hence, we must determine when *CI* is left next. As the input stream ι is piecewise smooth, there must be a time $t_1 > t_0$, such that it evolves continuously from now up to t_1. Due to its resolvability, the analog part must have a fixed point σ for this input and starting state s'. This fixed point also is a continuous function. Above, we demanded that the set $CI \subseteq \mathcal{I} \times n \cdot \mathcal{S}$ for which *Com* can idle is topologically open. Therefore, *CI*, restricted to the range of ι and σ for the interval $[t_0, t_1)$, is also open with respect to the induced subspace topology on $(\mathcal{I} \times n \cdot \mathcal{S}) \cap range((\iota, \sigma)|_{[t_0,t_1)})$ (subspace topologies are defined in Appendix B.2, Theorem B.4). Hence, constructing the inverse image of *CI* for the fixed point of *Ana* and the input stream up to t_1 yields a set I that is open w.r.t. the natural topology on $dom((\iota, \sigma)) = [t_0, t_1)$, since the input and the analog part's output are continuous functions up to t_1. As t_0 is in this set and the set is open w.r.t. $[t_0, t_1)$, we can conclude that there is a $\delta_0 > 0$ such that $[t_0, t_0 + \delta_0) \subseteq I$, i.e. *Com* can idle during the interval $[t_0, t_0 + \delta_0)$. We select this execution thread where *Com* idles on $[t_0, t_0 + \delta_0)$ from the set of nondeterministic alternatives.

Then on the interval $[t_0, t_0+\delta_0)$, the fixed point of *Ana* is a fixed point of *Cmp*, because *Com*, due to idling, and Lim_s, due to continuity of σ, are the identity there. Applying this argument inductively, we find a fixed point for *Cmp* on the interval $[0, \Sigma_{n=0}^{\infty}\delta_n)$ for every initial state s_0. □

If $\Sigma_{n=0}^{\infty}\delta_n$ diverges, we have constructed a proper fixed point of *Cmp*, since Theorem A.2 in Appendix A.1.1 allows us to transfer this result from finite time to infinite time. The actual output of *Cmp* for this fixed point is obtained by applying projection $Out^{\dagger}$ to it. Otherwise we have a *zeno* execution, i.e. the discrete part performs infinitely many discrete moves within a finite interval. Hence, it is not realizable. A sufficient condition for realizability is that there is a lower bound δ on the δ_i for all inputs and initial states. If the analog part is resolvable and the discrete part is realizable with respect to the analog part then the component delivers a reasonable, i.e. infinite and piecewise smooth, output for all reasonable inputs. In other words, the component is total. According to the principal idea given in [AH97] for *receptiveness*, we call a total component *receptive*. Note that execution threads where *Com* performs infinitely many discrete moves in a finite interval are eliminated by the type of the component only allowing piecewise smooth output.

3.2.5 A Note on Semantics

A very important characteristic of our semantic model is its uniform use of relations. Both, activities and the component itself, are time-guarded relations. Moreover, the discrete part is also a relation, but a relation without time and memory. This uniformity has two important consequences. First, it considerably simplifies the semantic

definition. Second, it allows us to apply the operators on hierarchic graphs introduced in the following section to compose relations. As we shall see in the following, these operators correspond to hierarchic system architecture specifications for the components and to hierarchic state-based specifications for the discrete part. A time extended variant of the operators for state-based specifications leads to hierarchic activity-based specifications for the analog part.

Using dense communication histories as the basis for component specification allows us to integrate hybrid machines with components that are specified in other formalisms. In particular this includes well-established description techniques from control theory where a component usually is a function from its inputs to its outputs, $\mathcal{I}^{\mathbb{R}_+} \to \mathcal{O}^{\mathbb{R}_+}$ without continuity restrictions in this case [Son90].

3.3 Hierarchic Graphs

This section first introduces the operators for an algebra of hierarchic graphs (Section 3.3.1). Then three relational models for this algebra, a *multiplicative*, an *additive* and a *time-extended additive* model, are given. The multiplicative model interprets the operators in a way that yields data-flow graphs (Section 3.3.2), the additive model interprets them in a way that results in control-flow graphs (Section 3.3.3) and the time extended additive model interprets them in a way that integrates data-flow aspects into contol-flow graphs (Section 3.3.4). These three models provide the formal foundation for HyACharts and HySCharts. Section 3.3.5 introduces a notion of refinement and relates it to the three models.

3.3.1 Syntax

A *hierarchic graph* consists of a set of *nodes* connected by a set of *arcs*. Each node has a name and, if it is not a leaf node, it again has associated a graph. For each node, the incoming and the outgoing arcs define the node's *interface*, i.e. its *type*. We use the textual notation $n : A \triangleright B$ to denote the node n with input interface A and output interface B. Its graphical representation is depicted in Figure 3.8. If we interpret A and B as sets (or types), n may be regarded as a mapping of elements of A to elements of B. However, note that there are other interpretations as well; we only define syntax here. The interpretations of the hierarchic graph syntax in Sections 3.3.2, 3.3.3 and 3.3.4 will define more details of n.

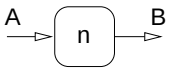


Figure 3.8: Graphical representation of a node $n : A \triangleright B$.

3.3.1.1 Operators on Nodes

In order to obtain graphs, we put nodes next to one another and connect them by using the following operators on relations: *sequential composition, visual attachment* and *feedback*. Their respective visual representation is given in Figure 3.9.

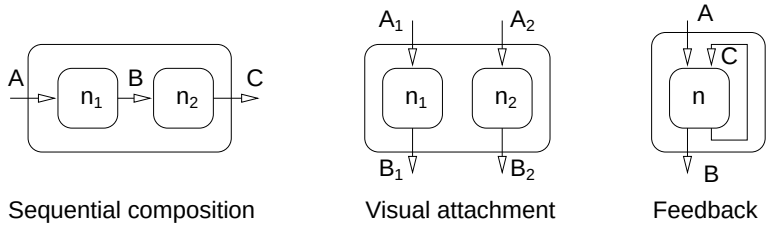


Figure 3.9: The composition operators.

Sequential composition. One basic way to connect two nodes is by *sequential composition,* i.e. as shown in Figure 3.9, left, by connecting the output of one node to the input of the other node, if they have the same type. Textually we denote this operator by the semicolon ;. Given $n_1 : A \rhd B$ and $n_2 : B \rhd C$ we define the composed node $n_1; n_2$ to have interface $A \rhd C$.

Regarding the nodes as computation units, Figure 3.9, left, says that the output produced by n_1 is directed to the input of n_2. The connection between n_1 and n_2 as well as the units n_1 and n_2 themselves are internal to $n_1; n_2$. In other words, $n_1; n_2$ does not only define a *connection* relation but also a *containment* relation.

Visual attachment. By *visual attachment* we mean that nodes and corresponding arcs are put one next to the other, as shown in Figure 3.9, middle. To obtain a textual representation for visual attachment, we need an attachment operator both on arcs and on nodes. We denote this operator by $\star$. Given two arcs A and B their visual attachment is expressed by $A \star B$. Given two nodes $n_1 : A_1 \rhd B_1$ and $n_2 : A_2 \rhd B_2$ their visual attachment is expressed as $n_1 \star n_2 : A_1 \star A_2 \rhd B_1 \star B_2$. Visual attachment also defines a containment relation. We say that n_1 and n_2 are contained in $n_1 \star n_2$.

In order to express loss of information (hiding, see below) it is convenient to have an arc E denoting the *absence* of any information. This arc should therefore be neutral for attachment, i.e. $A \star E = E \star A = A$.

Feedback. Sequential composition only allows us to connect the output of one node to the input of another. In particular, it cannot connect inputs and outputs of the same node or connect nodes with bidirectional communication. We therefore introduce a *feedback operator*, as shown in Figure 3.9, right. It allows us to

connect the rightmost output of a node to the rightmost input of the same node, if they have the same type. Given $n : A \star C \rhd B \star C$ we define $n \uparrow^C_{A,B}$: $A \rhd B$. Similar to sequential composition and visual attachment, feedback also introduces a containment relationship. We say that n and the feedback arc are contained in $n \uparrow^C_{A,B}$.

Nodes and arcs that are not built up from other nodes or arcs using the above operators are called *primitive*.

3.3.1.2 Connectors

Beside operators on nodes, we also need some operators on arcs (or predefined nodes), which we call *connectors*. We consider the following connectors: *identity, identification, ramification* and *transposition*. Their visual representation is given in Figure 3.10.

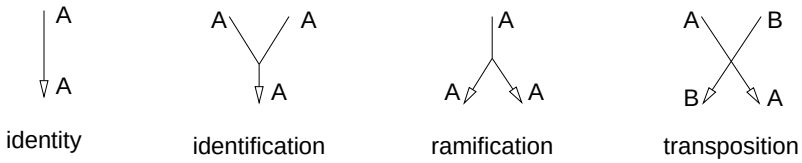


Figure 3.10: The connectors.

Identity. The identity connector I_A simply copies its input to the output. It has interface $A \rhd A$.

Identification. The *identification* connector $\vee^k_A$ joins k inputs together. Its interface is $A^k \rhd A$, where $A^k = A \star \ldots \star A$ stands for the k-fold attachment of A. For $k = 0$ we define $A^0 = E$, i.e. the neutral arc. $\vee^0_A : E \rhd A$ is the neutral element for identification. In Figure 3.10 binary identification is depicted.

Ramification. The *ramification* connector $\wedge^A_k$ copies the input information on k outputs. Hence $\wedge^A_k$ has interface $A \rhd A^k$. Figure 3.10 shows the binary case. The nullary case $\wedge^A_0 : A \rhd E$ expresses hiding; it is the neutral element for ramification.

Transposition. Finally the *transposition* connector ${}^A\mathsf{X}^B$ exchanges the inputs. Its interface is $A \star B \rhd B \star A$.

To be a precise formalization of our intuitive understanding of graphs, the above abstract operators and connectors have to satisfy a set of axioms, which intuitively express our visual understanding of graphs. These axioms correspond to *strict, symmetric, monoidal categories with feedback* and *bimonoid objects*, see e.g., [GBSS98,

GBSS99, Şte94, Şte00]. Some of the axioms are used and briefly explained in Chapter 4. [GSB98b] shows that the *multiplicative* and the *additive interpretations* of the operators and connectors are particularly relevant for computer science.

Example 3.3 As an example for a hierarchic graph and its corresponding textual representation, we consider the graph in Figure 3.11, left. Using the above basic

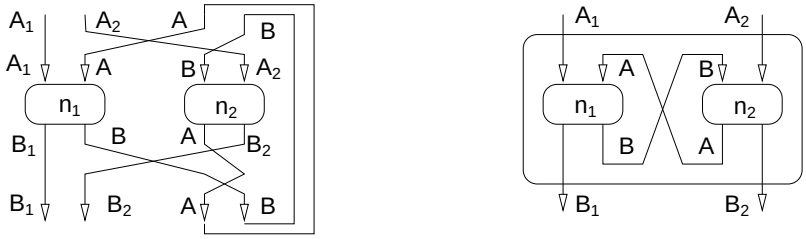


Figure 3.11: The symmetric feedback.

operators and connectors, it defines a *derived* composition operator, the *symmetric feedback*. If $n_1 : A_1 \star A \triangleright B_1 \star B$ and $n_2 : B \star A_2 \triangleright A \star B_2$ then $n_1 \circledast n_2$ has interface $A_1 \star A_2 \triangleright B_1 \star B_2$. Its simplified visual representation is given in Figure 3.11, right. The textual representation corresponds one-to-one to the visual representation in Figure 3.11, left:

$$\begin{aligned} n_1 \circledast n_2 \quad = \quad & (((\mathsf{I}_{A_1} \star {}^{A_2}\mathsf{X}^{A\star B});(n_1 \star n_2);(\mathsf{I}_{B_1} \star {}^{B\star A}\mathsf{X}^{B_2});\\ & (\mathsf{I}_{B_1\star B_2} \star {}^{B}\mathsf{X}^{A}))\uparrow^{B}_{A_1\star A_2\star A, B_1\star B_2\star A})\uparrow^{A}_{A_1\star A_2, B_1\star B_2} \end{aligned}$$

□

3.3.2 The Multiplicative Model

The *multiplicative model* is a model for *hierarchic data-flow graphs*. It is needed to define the semantics of HyACharts. The intuition behind hierarchic data-flow graphs is as follows. At any moment in time, *all* nodes of the graph are active and computing the output data based on their input data. A node receives the input data along a *tuple of input channels* and sends the computed data along a *tuple of output channels*. The arcs of the graph, i.e. the channels, forward the data to the other nodes in the graph. The nodes in a data-flow graph are also called *components*. The intended parallelism of nodes, input/output channels and branches of the connectors is obtained by interpreting the visual attachment $\star$ *multiplicatively* by the *(associative Cartesian) product* $\times$ and by defining the other operators and connectors consistently with this interpretation.

3.3.2.1 Arcs

We assume given a set of *channel types* $\mathcal{D} = \{D_i \mid i \in \mathbb{N}\}$, each defining the set of values which is allowed to flow along a channel. The input and the output *interface type* of a node, respectively, is then a flat product $A = A_1 \times \ldots \times A_n$ of channel types $A_i \in \mathcal{D}$, defined as follows:

$$A = \{()\} \text{ if } n = 0, \quad A = A_1 \text{ if } n = 1,$$
$$A = \{(x_1, \ldots, x_n) \mid x_1 \in A_1 \wedge \ldots \wedge x_n \in A_n\} \text{ if } n > 1$$

For arbitrary interface types $A = A_1 \times \ldots \times A_m$ and $B = B_1 \times \ldots \times B_n$ we extend the above product definition as follows:

$$\begin{array}{lll} A \times \{()\} & = \{()\} \times A = A & \\ A \times B & = \{(x_1, \ldots, x_{m+n}) \mid & x_1 \in A_1 \wedge \ldots \wedge x_m \in A_m \wedge \\ & & x_{m+1} \in B_1 \wedge \ldots \wedge x_{m+n} \in B_n\} \text{ if } m, n > 1 \end{array}$$

Hence, the empty interface $\{()\}$ is the neutral arc E. The *left* and *right projections* $p.$ and $q.$ are given below:

$$p. : A_1 \times \ldots \times A_m \times B_1 \ldots \times B_n \to A_1 \times \ldots \times A_m,$$
$$p.(a_1, \ldots, a_m, b_1, \ldots, b_n) = (a_1, \ldots, a_m)$$
$$q. : A_1 \times \ldots \times A_m \times B_1 \ldots \times B_n \to B_1 \times \ldots \times B_n,$$
$$q.(a_1, \ldots, a_m, b_1, \ldots, b_n) = (b_1, \ldots, b_n)$$

They uniquely define a pairing function $(.,.)$ such that for any C, $f = (f_1, \ldots, f_m) \in C \to A$ and $g = (g_1, \ldots, g_n) \in C \to B$ it holds that: $(f, g) = (f_1, \ldots, f_m, g_1, \ldots, g_n)$. By definition, the product is associative and has as neutral element E.[8] The unique existence of projections is characteristic of data-flow graphs.

Time models. In data-flow graphs the main concern is the data *flow*. To define and analyze this flow, we need to observe the information exchanged along each channel over *time*. As different time models will be used throughout the thesis, namely the non-negative real numbers $\mathbb{R}_+$, the natural numbers including zero $\mathbb{N}$ and certain classes of subintervals thereof, we overload the definition of multiplicative hierarchic graphs for any of these time models $\mathcal{T}$.

As motivated in Section 3.2.1, for HyCharts we use time model $\mathcal{T} = \mathbb{R}_+$. In some proofs, however, we will also use finite time models $\mathcal{T} = [0, t)$, for $t > 0$. Later in the thesis we will consider the implementation of (parts of) hybrid systems with an underlying discrete time grid. In that context, the natural numbers including zero and subsets containing a sequence of numbers $\{0, \ldots, k\}$, $k \in \mathbb{N} \setminus \{0\}$, are a second choice for the time axis. We also call such sets $\{0, \ldots, k\}$ *intervals (of natural*

[8] In fact, associativity is the motivation for explicitly giving a definition for the Cartesian product here. Other definitions in the literature, for instance the definition in [Eng89], are only associative w.r.t. an isomorphism.

numbers). In any case, the data exchanged along a channel with type A over time defines a mapping $a \in \mathcal{T} \to A$. The set of mappings $\mathcal{T} \to A$ is also denoted by $A^{\mathcal{T}}$. If the used time model is the real axis or a subinterval thereof, we usually require that the mapping is piecewise smooth and piecewise Lipschitz continuous, as defined in Section 3.2.3. This is assumed in all the following definitions of the operators and arcs. Thus, $A^{\mathcal{T}}$ denotes the set of piecewise smooth and piecewise Lipschitz continuous mappings in case of such time models.[9] Such a restricted mapping is called a *dense communication history* (or *dense stream*) and its corresponding type a *dense communication history type*. If the natural numbers (or a subinterval thereof) are used as time model, no further restrictions are required. The resulting mapping is called a *discrete-time communication history* (or *discrete-time stream*) and its type a *discrete-time communication history type*. Dense or discrete-time communication history types, respectively, are used to interpret the *primitive arcs* of a data-flow graph.

A reasonable assumption which leads to a model with very nice properties, is that data-flows are *time synchronous*, i.e. that time flows in the same way for each channel and each component. In this case, the history type of a component's interface $(A_1 \times \ldots \times A_m)^{\mathcal{T}}$ is equal to the product $A_1^{\mathcal{T}} \times \ldots \times A_m^{\mathcal{T}}$ of the histories types of its channels (up to isomorphism).

3.3.2.2 Nodes and Operators

As indicated in Section 3.2.1 for time model $\mathbb{R}_+$, we describe the behavior of a system component by an *input/output relation*, i.e. by a relation between the histories of its input channels and the histories of its output channels. The relation must be total in the input histories and time guarded.[10] To emphasize totality we also use the functional notation $n \in A^{\mathcal{T}} \to \mathcal{P}(B^{\mathcal{T}})$ for such relations $n \subseteq A^{\mathcal{T}} \times B^{\mathcal{T}}$ in this thesis. These relations *interpret* the *nodes* of the data-flow graphs. To simplify notation, we use the same name (or symbol) for a node (or operator) and its associated relation (or relational operator) in the following. Note, however, that the names and symbols are syntactic entities whereas the relations and relational operators are semantic entities.

The Node Operators:

Multiplicative sequential composition. The multiplicative interpretation of sequential composition is the usual sequential composition of relations. It allows passing of the data from one component to another component in a linear way. Given two relations

$$n_1 \subseteq A^{\mathcal{T}} \times B^{\mathcal{T}}, \quad n_2 \subseteq B^{\mathcal{T}} \times C^{\mathcal{T}}$$

[9] Note that the definitions are the same for other kinds of continuity requirements.

[10] Time guardedness w.r.t. a discrete time model is similar to the definition in Section 3.2.1 for continuous time, but only regards intervals of natural numbers.

we define their multiplicative sequential composition $n_1 ;_\times n_2$ as follows:

$$\begin{aligned} n_1 ;_\times n_2 &\subseteq A^{\mathcal{T}} \times C^{\mathcal{T}} \\ n_1 ;_\times n_2 &= \{(a,c) \mid \exists b \in B^{\mathcal{T}}.\ (a,b) \in n_1 \ \wedge\ (b,c) \in n_2\} \end{aligned}$$

$n_1 ;_\times n_2$ is total in its input and time guarded if n_1 and n_2 are total and time guarded as well.

Multiplicative visual attachment. The multiplicative visual attachment of two components yields a new component such that *both* constituents are active simultaneously, i.e. each constituent has its own control-flow. As this is similar to parallel composition (or *and*-composition) in Statecharts, we also refer to multiplicative visual attachment as *parallel composition*. The interface of the product has to reflect this fact. Given two relations

$$n_1 \subseteq {A_1}^{\mathcal{T}} \times {B_1}^{\mathcal{T}}, \quad n_2 \subseteq {A_2}^{\mathcal{T}} \times {B_2}^{\mathcal{T}}$$

we define their product $n_1 \times n_2$ as follows:

$$\begin{aligned} n_1 \times n_2 &\subseteq ({A_1}^{\mathcal{T}} \times {A_2}^{\mathcal{T}}) \times ({B_1}^{\mathcal{T}} \times {B_2}^{\mathcal{T}}) \\ n_1 \times n_2 &= \{((a_1,a_2),(b_1,b_2)) \mid (a_1,b_1) \in n_1 \ \wedge\ (a_2,b_2) \in n_2\} \end{aligned}$$

The visual notation for $n_1 \times n_2$ is given in Figure 3.12. $n_1 \times n_2$ is total in its

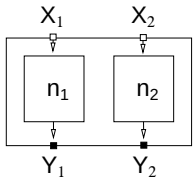


Figure 3.12: The multiplicative interpretation.

input and time guarded if n_1 and n_2 are.

Multiplicative feedback. The multiplicative feedback allows the passing of the output of a component back to its input. In our hybrid machine model depicted in Figure 3.4, middle, this constuct is used to add the memory to our components. Given a relation

$$n \subseteq (A^{\mathcal{T}} \times C^{\mathcal{T}}) \times (B^{\mathcal{T}} \times C^{\mathcal{T}})$$

we define the new relation $n \uparrow_\times^C$ as below:

$$\begin{aligned} n \uparrow_\times^C &\subseteq A^{\mathcal{T}} \times B^{\mathcal{T}} \\ n \uparrow_\times^C &= \{(a,b) \mid \exists c.\ ((a,c),(b,c)) \in n\} \end{aligned}$$

$n \uparrow_\times^C$ is time guarded and guaranteed to be total in the input channel histories $A^{\mathcal{T}}$, if n is time guarded and its output channel with type C up to time $t+\delta \in \mathcal{T}$

is completely determined by its input up to time $t \in \mathcal{T}$ on the input channel with channel type C and by the input on the other input channels up to time $t + \delta$. In other words, its output on the output channel with type C reacts with a delay $\delta > 0$ to the input on the input channel with type C. We also say that n is *strongly* time guarded on the feedback channel with channel type C. (See [Bro97b, SRS99] for proofs with dense time models and [GR95] for a proof with discrete time model.)

We also allow feedback of components without delay on their feedback channel, as long as the feedback results in a time-guarded component that is total in its input. In this case, the designer must explicitly prove totality and time guardedness of the resulting component. For instance, such a proof is given for the hybrid computation model in Section 3.2.4 and Appendix A.1.[11]

Note that an infinitely small delay on a feedback channel, as caused by the *Lim* component in the hybrid machine model, is not guaranteed to be sufficient in general to conclude that the resulting component is total and time guarded. The proofs for totality and time guardedness given in [MS97, Bro97b] and [SRS99] can not be used in this case, because they all require a finitely large delay $\delta > 0$.

The Connectors:

Multiplicative identity. We interpret the identity connector $\mathsf{I}_A : A \rhd A$ by the identity relation I_A which simply copies the input to the output:

$$\mathsf{I}_A \subseteq A^{\mathcal{T}} \times A^{\mathcal{T}}, \qquad \mathsf{I}_A = \{(a, a) \mid a \in A^{\mathcal{T}}\}$$

Multiplicative identification. The identification connectors $\curlyvee_A^k : A^k \rhd A$ are interpreted by the multiplicative identification relations $\curlyvee_A^k$. They allow us to identify k equal copies of elements $a \in A^{\mathcal{T}}$:

$$\curlyvee_A^k \subseteq (A^k)^{\mathcal{T}} \times A^{\mathcal{T}}, \qquad \curlyvee_A^k = \{(a^k, a) \mid 0 < k \ \wedge \ a \in A^{\mathcal{T}}\}$$

Note that $(A^k)^{\mathcal{T}} = (A^{\mathcal{T}})^k$, as we are using a time-synchronous setting. The neutral element $\curlyvee_A^0$ is defined by $\curlyvee_A^0 = \{((), a) \mid a \in A^{\mathcal{T}}\}$.

Unlike the other connectors, multiplicative identification is not total. If the input histories differ, $\curlyvee_A^k$ provides no output. Thus, multiplicative identification introduces a kind of synchronization condition on its input which is similar to synchronization in Petri nets. We will not need multiplicative identification in this thesis.

[11] Informally, for the hybrid computation model totality holds, because of two main reasons. First, circular dependencies are broken up by the *Lim* component whenever *Com* performs a move. Second, between these moves resolvability of *Ana* guarantees that an output exists (see also Section 3.2.4).

Multiplicative ramification. The ramification connectors $\wedge_k^A : A \rhd A^k$ are interpreted by the multiplicative ramification relations $\wp_k^A$. They allow us to make k equal copies of the input:

$$\wp_k^A \subseteq A^{\mathcal{T}} \times (A^k)^{\mathcal{T}}, \qquad \wp_k^A = \{(a, a^k) \mid 0 < k \ \wedge \ a \in A^{\mathcal{T}}\}$$

The neutral element $\wp_0^A$ is defined by $\wp_0^A = \{(a, ()^{\mathcal{T}}) \mid a \in A^{\mathcal{T}}\}$.

Multiplicative transposition. The transposition connectors ${}^A\mathsf{X}^B : A \star B \rhd B \star A$ are interpreted by the multiplicative transposition relations ${}^A\mathsf{X}^B$ which allow us to commute the position of the elements in the input tuple.

$$\begin{aligned} {}^A\mathsf{X}^B &\subseteq (A^{\mathcal{T}} \times B^{\mathcal{T}}) \times (B^{\mathcal{T}} \times A^{\mathcal{T}}), \\ {}^A\mathsf{X}^B &= \{((a,b),(b,a)) \mid a \in A^{\mathcal{T}} \ \wedge \ b \in B^{\mathcal{T}}\} \end{aligned}$$

Apart from multiplicative identification all the connectors are total in their input and time guarded. Multiplicative identification will not be needed further in the thesis. When we argue over multiplicative hierarchic graphs in the following, we do not consider identification, but only the other operators and arcs. Thus, as long as all primitive nodes and every feedback result in total, time-guarded relations, a multiplicative hierarchic graph also defines a total, time-guarded relation.

Unless otherwise mentioned, we always interpret the operators and connectors w.r.t. time model $\mathcal{T} = \mathbb{R}_+$ in the context of HyCharts.

Example 3.4 (Graph for the hybrid machine model.) The hierarchic graph for the hybrid computation model depicted in Figure 3.4, middle, corresponds to the following definition for the semantics *Cmp* of a hybrid machine for initial state s:

$$Cmp(s) \ = \ ((\wp_2^{\mathcal{I}} \times \mathsf{I}_{n\cdot\mathcal{S}}) \ ;_\times \ (\mathsf{I}_{\mathcal{I}} \times Com^\dagger) \ ;_\times \ Ana \ ;_\times \ \wp_2^{n\cdot\mathcal{S}} \ ;_\times \ (Out^\dagger \times Lim_s)) \uparrow_\times^{n\cdot\mathcal{S}}$$

where *Com*, *Ana*, *Out*, *Lim* and the sets $\mathcal{I}$ and $n \cdot \mathcal{S}$ are as described in Section 3.2. Note that the types of the connectors can also be inferred from the types of the other relations. Therefore, they are not always given explicitly in the textual representations of hierarchic graphs in this thesis. Expanding the definitions of the graph operators yields the formula given in Section 3.2.4 for *Cmp*. □

3.3.3 The Additive Model

The *additive model* is a model for *hierarchic, sequential control-flow graphs*. It is needed to define the semantics of the discrete part in HySCharts. The intuition behind hierarchic, sequential control-flow graphs is as follows. Each node in the graph corresponds to a unit which performs a computation. Only one node in the graph can be active at any point in time. A node receives the control on one of its *entry points*, performs its computation and gives the control back on one of its *exit points*. Entry

and exit points are *disjoint*, i.e. control can only be received or given by *one* of them. The arcs of the graph forward the control to the other nodes of the graph. The intended disjointedness of nodes, entry/exit points and branches of the connectors is obtained by interpreting visual attachment *additively* as *disjoint sum* (see below) and by defining the other operators and connectors consistently with this interpretation. From an external view, all computation is the graph happens instantaneously. Essentially a control-flow graph can be regarded as defining a state-transition relation.

3.3.3.1 Arcs

In the additive interpretation the *control* which is passed between the nodes in a control-flow graph is defined to be an element of the data-state space, $s \in \mathcal{S}$. Thus, we consider a set $\mathcal{S}$, the data-state space, as introduced in Section 3.2.1, to be given. The node which has the data state is active and can modify it. Each arc in a control-flow graph is interpreted as forwarding values from $\mathcal{S}$. The visual attachment of n arcs is interpreted as forwarding values from the n fold disjoint sum of $\mathcal{S}$ (see Figure 3.13).

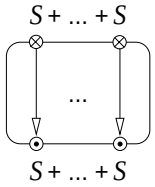


Figure 3.13: Visual attachment of arcs in the additive interpretation.

The n fold *disjoint sum* $\mathcal{S} + \ldots + \mathcal{S}$, abbreviated by $n \cdot \mathcal{S}$, is defined as follows:

$$0{\cdot}\mathcal{S} = \emptyset, \quad 1{\cdot}\mathcal{S} = \mathcal{S}, \quad n{\cdot}\mathcal{S} = \{(k, s) \mid 0 < k \leq n \wedge s \in \mathcal{S}\}, \text{ if } n > 1$$

We take the *empty set* as the interpretation of the *neutral arc* E. Set $n \cdot \mathcal{S}$ is also called the *program-state space*. Its elements (k, s) consist of the *control state* k and the data state s. The control state indicates that s stems from the k-th summand in the n fold sum $n \cdot \mathcal{S}$. The graphical analogy is that the data state is received on the k-th of n visually attached arcs. This reflects that control information is encoded in the interface of nodes and arcs in a control-flow graph: The entry and exit points of a node correspond to control states and the arcs, which connect exit and entry points, therefore lead from one control state to the next. Note that not all control states (or entry/exit points) are visible at the interface of a control-flow graph. As we will see in Section 3.5.1 the externally visible entry/exit points define the interface of the discrete part of a HySChart, i.e. of its state-transition relation.

The disjoint sum of program-state spaces is defined by the following equation: $m{\cdot}\mathcal{S} + n{\cdot}\mathcal{S} = (m + n){\cdot}\mathcal{S}$. In a graph, it corresponds to the visual attachment of n visually

attached arcs with m visually attached arcs. From the left and right summands $m{\cdot}\mathcal{S}$ and $n{\cdot}\mathcal{S}$ there are two canonical functions into the sum $(m+n){\cdot}\mathcal{S}$, called the *left injection* $l.$ and the *right injection* $r.$. They *inject* elements from the summands into the sum such that one can recover their original source. Their definition is as follows:

$$\begin{aligned} l. &: m{\cdot}\mathcal{S} \to (m+n){\cdot}\mathcal{S}, \quad l.(k,s) = (k,s) \\ r. &: n{\cdot}\mathcal{S} \to (m+n){\cdot}\mathcal{S}, \quad r.(k,s) = (k+m,s) \end{aligned}$$

It is easy to see that the sum is associative and the neutral element is $0{\cdot}\mathcal{S}$. In the following we will often refer to the program state as merely the state.

3.3.3.2 Nodes and Operators

A node $n : A \triangleright B$ of a control-flow graph is *interpreted* as a relation $n \subseteq (\mathcal{I} \times k{\cdot}\mathcal{S}) \times l{\cdot}\mathcal{S}$ between the *current external input*, the *current state* and the *next state*. $\mathcal{I}$ denotes the input space, as defined in Section 3.2.1. Upon receiving the current state, the nodes performs its computation and determines the next state, depending on the received state and the current input. In addition, we consider an external input here, because the sequential machines defined by the relations are allowed to communicate with their environment. They may receive input and produce output. The output space simply is a projection of the data-state space. The definition of the operators below ensures that all nodes receive the same input. Therefore, by convention no arc is drawn to denote the external input to a node.

Note that in order to simplify notation, we use the same name for the node, which is a syntactic entity, and its associated relation, which is a semantic entity. In the following we denote arbitrary program-state spaces $m_{A_i} \cdot \mathcal{S}$, $m_{B_i} \cdot \mathcal{S}$ and $m_C \cdot \mathcal{S}$ over data-state space $\mathcal{S}$ by A_i, B_i and C.

The Node Operators:

Additive sequential composition. The additive sequential composition of two nodes

$$n_1 \ \subseteq (\mathcal{I} \times A) \ \times \ B, \qquad n_2 \ \subseteq (\mathcal{I} \times B) \ \times \ C$$

yields a new node, $n_1 \,;_+ \, n_2$, which is defined as expected:

$$\begin{aligned} n_1 \,;_+ \, n_2 \ &\subseteq (\mathcal{I} \times A) \ \times \ C \\ n_1 \,;_+ \, n_2 \ &= \{((x,a),c) \mid \exists b \in B.\ ((x,a),b) \in n_1 \ \wedge \ ((x,b),c) \in n_2\} \end{aligned}$$

Note that n_2 gets the same external input as n_1.

As example for additive sequential composition let us use an analogy: If we think of the nodes as states in an automata diagram and of the arcs as transitions between them, additive sequential composition of the nodes expresses that all outgoing transitions of n_1 lead to n_2 and that n_2 can only be entered via n_1. Note, however, that this is just an analogy. Unlike to the intuitive interpretation

of automata diagrams, no time passes in a node of a control-flow graph. Instead, its computation happens instantaneously. It is a state-transition relation without a notion of time.

Additive visual attachment. The additive visual attachment of two nodes

$$n_1 \ \subseteq (\mathcal{I} \times A_1) \ \times \ B_1, \qquad n_2 \ \subseteq (\mathcal{I} \times A_2) \ \times \ B_2$$

yields, as *or*-composition in Statecharts, a new node $n_1 + n_2$, such that computation is performed either by n_1 or by n_2. Note that the interface of the sum reflects this fact.

$$\begin{aligned} n_1 + n_2 \ &\subseteq (\mathcal{I} \times (A_1 + A_2)) \ \times \ (B_1 + B_2) \\ n_1 + n_2 \ &= \{((x, l.a), l.b) \mid ((x, a), b) \in n_1\} \ \cup \ \{((x, r.a), r.b) \mid ((x, a), b) \in n_2\} \end{aligned}$$

We also call the additive visual attachment of nodes the *(disjoint) sum* of them. The visual notation of $n_1 + n_2$ is given in Figure 3.14. The meaning of $(n_1 + n_2)(x, l.a)$ can be intuitively understood as follows. Receiving the tuple $(x, l.a)$, the sum uses the control information $l.$ to select the corresponding relation n_1. This relation is then applied to (x, a) to obtain the next state b. Finally, the control information is added to the output again and $l.b$ is obtained. The situation is symmetric when tuple $(x, r.a)$ is received.

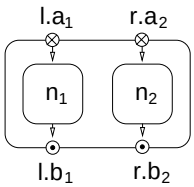


Figure 3.14: Visual attachment of nodes in the additive interpretation.

In the above analogy of automata diagrams, additive visual attachment expresses that two distinct states not coupled with each other by transitions are grouped to a hierarchic state. Entering the hierarchic state means that exactly one of its substates is entered, namely the one specified by the control information l or r.

Additive feedback. The additive feedback is more tricky and it allows the construction of loops. As in programming, feedback has to be used with care in order to ensure termination. Given a relation

$$n \ \subseteq (\mathcal{I} \times (A + C)) \ \times \ (B + C)$$

we define the relation $n \uparrow_+^C$ as follows: The state is received on A, computation is performed and the next state is either given directly on B or after an arbitrary number of further computations in which the state loops along C. Formally:

$$n \uparrow_+^C \ \subseteq (\mathcal{I} \times A) \ \times \ B$$
$$n \uparrow_+^C \ = n_{l,l} \ \cup \ n_{l,r} \, ; \, n_{r,r}^* \, ; \, n_{r,l}$$

where m^* denotes the arbitrary but finite iteration of relation $m \subseteq (\mathcal{I} \times C) \times C$, i.e. $m^* = \bigcup_{i \geq 0} m^i$, where $m^{i+1} = m^i \, ;_+ \, m$ and $m^0 = \{((x, c), c) \mid c \in C \ \wedge \ x \in \mathcal{I}\}$. Relations $n_{i,j}$ are defined for $i, j \in \{l, r\}$ as below:

$$n_{i,j} = \{((x, s), s') \mid ((x, i.s), j.s') \in n\}$$

In this definition $l.$ and $r.$ are the injections corresponding to A and C for the input and to B and C for the output.

Using the analogy of automata diagrams again, additive feedback expresses that the rightmost outgoing transition of a state is leading back to itself. Hence, the state can be reentered via this transition several times before it is left via another transition.

Note that these interpretations of the composition operators result in a purely sequential nature of control-flow graphs and hence of the discrete part of HySCharts. No computations can be performed in parallel in such a control-flow graph.

The Connectors:

Additive identity. The additive identity i_A is defined as expected:

$$\mathsf{i}_A \ \subseteq (\mathcal{I} \times A) \times A, \qquad \mathsf{i}_A \ = \{((x, a), a) \mid a \in A \ \wedge \ x \in \mathcal{I}\}$$

It simply passes control on without modification.

If we extend our analogy of automata diagrams to hierarchic automata diagrams such as Statecharts (or, even closer, to ROOMcharts [SGW94]), additive identity may occur when a transition is split into segments at the boundary of a hierarchic state. The second part of the transition can be associated with the identity connector because it only forwards control.

Additive identification. The additive identification ${}_k\!\succ\!\bullet_A$ forgets the entry point on which it gets the state:

$${}_k\!\succ\!\bullet_A \ \subseteq (\mathcal{I} \times k\, A) \times A,$$
$${}_k\!\succ\!\bullet_A \ = \{((x, i.a), a) \mid 0 < i \leq k \ \wedge \ a \in A \ \wedge \ x \in \mathcal{I}\}$$

where $k\, A = k\, (m_A \cdot \mathcal{S}) = (k \cdot m_A) \cdot \mathcal{S}$. The neutral element ${}_0\!\succ\!\bullet_A$ is defined by ${}_0\!\succ\!\bullet_A = \emptyset$.

In the automata diagram analogy, additive identification corresponds to the so called *junction connectors* in Statecharts. For instance, they can be used to join transitions when several transitions have a common destination.

Additive ramification. The additive ramification ${}_A\bullet\!\!\prec_k$ nondeterministically forwards the state on one of its exit points:

$$\begin{aligned} {}_A\bullet\!\!\prec_k &\subseteq (\mathcal{I} \times A) \times k\,A, \\ {}_A\bullet\!\!\prec_k &= \{((x,a), i.a) \mid 0 < i \leq k \wedge a \in A \ \wedge \ x \in \mathcal{I}\} \end{aligned}$$

The neutral element ${}_A\bullet\!\!\prec_0$ is defined by ${}_A\bullet\!\!\prec_0 = \emptyset$.

In the automata diagram analogy, additive ramification is similar to the so-called *condition connectors* in Statecharts. These are typically employed if several transitions emerging from the same state are triggered by the same event. Note, however, that ramification only models the branching of the control flow. It does not contain conditions, but is nondeterministic.

Additive transposition. The additive transposition ${}^B_A\!\!\diagup\!\!\diagdown$ commutes the entry point information:

$$\begin{aligned} {}^B_A\!\!\diagup\!\!\diagdown &\subseteq (\mathcal{I} \times (A + B)) \times (B + A), \\ {}^B_A\!\!\diagup\!\!\diagdown &= \{((x, l.a), r.a) \mid a \in A \ \wedge \ x \in \mathcal{I}\} \cup \{((x, r.b), l.b) \mid b \in B \ \wedge \ x \in \mathcal{I}\} \end{aligned}$$

This means that the state is passed on along the right exit point if it was received on the left entry point and vice versa.

In the automata diagram analogy, additive transposition occurs in cases where transitions in the diagram intersect each other.

3.3.4 The Time-Extended Additive Model

The time-extended additive model is needed to compose (adapted) activities in our formalism. The composed activities in turn define the analog part of a HySChart. Similar to the nodes in multiplicative hierarchic graphs, activities are time-guarded relations over streams, but unlike nodes in multiplicative graphs they need not be total.

The time-extended additive model is a model for *time-extended control-flow graphs*. It mixes data-flow aspects into control-flow graphs. The intuition here is similar to control-flow graphs and can be described as follows. Unlike the additive model which only regards instantaneous computation without a notion of time, the time-extended additive model regards ongoing computation over time intervals. During each interval only one node in a visual attachment of nodes is active and permanently computes output data from its input data. Sequentially composed nodes are active simultaneously. An active node receives the input data at one of its *entry points* and gives the output data at one of its *exit points*. Entry and exit points are *disjoint*, i.e. data can only be received or given at *one* of them. The arcs of the graph forward the data to the other nodes of the graph. The intended disjointedness of nodes and data-flow via entry/exit points is obtained by interpreting visual attachment *additively*

as *time-extended disjoint sum* (see below). As far as the other operators and connectors are concerned, we will only define those which are actually needed for the composition of activities in the HyChart framework.

Time-extended control-flow graphs can be regarded as defining a state-transition relation that is extended in time. The time extension is performed in a way which provides that it does not introduce discontinuities in the output data which are not originated by an active node or by discontinuities in the input data.

3.3.4.1 Arcs

As with the additive model, the data passed at every point in time on primitive arcs between the nodes is an element of the data-state space $\mathcal{S}$. As data now flows over time, the type of primitive arcs here is $\mathcal{S}^{\mathcal{T}}$, i.e. the set of functions from $\mathcal{T}$ to $\mathcal{S}$, where $\mathcal{T}$ is one of the time models introduced in Section 3.3.2.1. The visual attachment of n arcs is interpreted as transmitting elements from the n fold time-extended disjoint sum of $\mathcal{S}^{\mathcal{T}}$. This sum $\mathcal{S}^{\mathcal{T}} + \ldots + \mathcal{S}^{\mathcal{T}}$ is defined by:

$$\mathcal{S}^{\mathcal{T}} + \ldots + \mathcal{S}^{\mathcal{T}} = (n \cdot \mathcal{S})^{\mathcal{T}}$$

where $n \cdot \mathcal{S}$ is the program-state space as in Section 3.3.3.1. Correspondingly, $(n \cdot \mathcal{S})^{\mathcal{T}}$ is called the *time-extended program-state space*. Similar to the untimed case, the sum of time-extended program-state spaces is defined as $(n \cdot \mathcal{S})^{\mathcal{T}} + (m \cdot \mathcal{S})^{\mathcal{T}} = ((m+n) \cdot \mathcal{S})^{\mathcal{T}}$. Left and right injections are defined by pointwise extension of the untimed injections:

$$\begin{aligned} \mathbf{l}. &: (m \cdot \mathcal{S})^{\mathcal{T}} \to ((m+n) \cdot \mathcal{S})^{\mathcal{T}}, \quad \forall t \in \mathcal{T}.\ (\mathbf{l}.s)(t) = l.(s(t)) \\ \mathbf{r}. &: (n \cdot \mathcal{S})^{\mathcal{T}} \to ((m+n) \cdot \mathcal{S})^{\mathcal{T}}, \quad \forall t \in \mathcal{T}.\ (\mathbf{r}.s)(t) = r.(s(t)) \end{aligned}$$

The time-extended sum is associative and the neutral element is $(0 \cdot \mathcal{S})^{\mathcal{T}}$.

3.3.4.2 Nodes and Operators

Corresponding to the untimed case, a node $n : A \rhd B$ of a time-extended additive hierarchic graph is *interpreted* as a time-guarded relation $n \subseteq (\mathcal{I}^{\mathcal{T}} \times (k \cdot \mathcal{S})^{\mathcal{T}}) \times (\ell \cdot \mathcal{S})^{\mathcal{T}}$ between the external input stream in $\mathcal{I}^{\mathcal{T}}$, the received state-stream in $(k \cdot \mathcal{S})^{\mathcal{T}}$ and the produced state-stream in $(\ell \cdot \mathcal{S})^{\mathcal{T}}$. Based on the evolution of the external input stream and the received state-stream the node computes the evolution of the produced state-stream. As in the untimed case, the definition of the operators below ensures that all nodes receive the same input. Therefore, by convention no arc is drawn to denote the external input $\mathcal{I}^{\mathcal{T}}$ to a node. The external input is needed to allow the analog part in our hybrid communication model to communicate with an (external) environment.

In the following definitions we denote arbitrary program-state spaces $(m_{A_i} \cdot \mathcal{S})$, $(m_{B_i} \cdot \mathcal{S})$ and $(m_C \cdot \mathcal{S})$ over data-state space $\mathcal{S}$ by A_i, B_i and C. Again, we use the same name for the node, which is a syntactic entity, and its associated relation, which is a semantic entity.

The Node Operators:

Time-extended additive sequential composition. The time-extended additive sequential composition of two nodes

$$n_1 \subseteq (\mathcal{I}^{\mathcal{T}} \times A^{\mathcal{T}}) \times B^{\mathcal{T}}, \qquad n_2 \subseteq (\mathcal{I}^{\mathcal{T}} \times B^{\mathcal{T}}) \times C^{\mathcal{T}}$$

yields, a new node $n_1 ;_+ n_2$, which is defined as expected:

$$\begin{aligned} n_1 ;_+ n_2 &\subseteq (\mathcal{I}^{\mathcal{T}} \times A^{\mathcal{T}}) \times C^{\mathcal{T}} \\ n_1 ;_+ n_2 &= \{((x,a),c) \mid \exists b \in B^{\mathcal{T}}.\ ((x,a),b) \in n_1 \ \wedge\ ((x,b),c) \in n_2\} \end{aligned}$$

Apart form the input stream x, this coincides with multiplicative sequential composition.

Time-extended additive visual attachment. The time-extended additive visual attachment of two nodes

$$n_1 \subseteq (\mathcal{I}^{\mathcal{T}} \times A_1^{\mathcal{T}}) \times B_1^{\mathcal{T}}, \qquad n_2 \subseteq (\mathcal{I}^{\mathcal{T}} \times A_2^{\mathcal{T}}) \times B_2^{\mathcal{T}}$$

yields a new node $n_1 + n_2$ for which the time axis $\mathcal{T}$ for each input in $A_1^{\mathcal{T}} + A_2^{\mathcal{T}}$ can be partitioned into intervals such that during each interval either n_1 or n_2 is active:

$$\begin{aligned} n_1 + n_2 &\subseteq (\mathcal{I}^{\mathcal{T}} \times (A_1^{\mathcal{T}} + A_2^{\mathcal{T}})) \times (B_1^{\mathcal{T}} + B_2^{\mathcal{T}}) \\ n_1 + n_2 &= \{((x,a),b) \mid \forall \delta \in \mathit{Int} \cap \mathcal{T}. \\ &\qquad (\exists a'.\, a|_\delta = \mathbf{l}.a'|_\delta \Rightarrow \\ &\qquad\quad \exists b'.\, ((x,a'),b')|_\delta \in n_1|_\delta \ \wedge\ b|_\delta = \mathbf{l}.b'|_\delta\,) \\ &\qquad \wedge \\ &\qquad (\exists a'.\, a|_\delta = \mathbf{r}.a'|_\delta \Rightarrow \\ &\qquad\quad \exists b'.\, ((x,a'),b')|_\delta \in n_2|_\delta \ \wedge\ b|_\delta = \mathbf{r}.b'|_\delta\,)\ \} \end{aligned}$$

where *Int* denotes the set of all left closed and right open intervals over $\mathbb{R}_+$. $\mathit{Int} \cap \mathcal{T}$ therefore are all such intervals w.r.t. the used time model. The definition expresses that for each interval δ in which the received state-steam a permanently is in A_i the output state-stream b is determined by n_i, $i \in \{1,2\}$.[12] In other words, the sum uses the control information in a to select the active node and adds the control information to the node's output again. In contrast to a pointwise time-extension of the untimed sum operator, this interval-based definition ensures that the selection of an output stream b cannot be changed nondeterministically at any point in time but only when the control information in the received state-stream changes from A_1 to A_2 or vice versa. Thus, this definition based on intervals does not introduce discontinuities in b as long as there

[12] For other intervals δ selected by the universal quantification, the formula does not restrict the output b, because of the implications in the formula.

is no discontinuity in a or the discontinuity is caused by the active subnode. This is required for the switching between (primitive) activities in the analog part of HySCharts.

In the relevant cases for HyCharts, we fully apply associativity of $+$. This results in a specialization of the above definition which may be easier to understand and is closer to our applications. Namely, we can directly define the time-extended disjoint sum of ℓ primitive nodes $n_j \in \mathcal{I}^{\mathcal{T}} \times \mathcal{S}^{\mathcal{T}} \times \mathcal{S}^{\mathcal{T}}$ for $1 \leq j \leq \ell$ as follows:

$$+_{j=1}^{\ell} n_j \;=\; \{\, ((x,(k,a)),(k,b)) \in \mathcal{I}^{\mathcal{T}} \times (\ell \cdot \mathcal{S})^{\mathcal{T}} \times (\ell \cdot \mathcal{S})^{\mathcal{T}} \mid \\ \forall \delta \in \mathit{Int} \cap \mathcal{T}, m \in \{1,\ldots,\ell\}.\; k|_\delta = m^\dagger|_\delta \Rightarrow ((x,a),b)|_\delta \in n_m|_\delta \,\}$$

where $m^\dagger$ is the extension of m to a constant function over $\mathcal{T}$. The tuple (k,a) consists of the control-state stream k which defines the control state at each moment in time and the data-state stream a which gives the data state at each moment in time (see also Figure 3.6). The tuple (k,b) consists of the same control-state stream k and the data-state stream b computed by the sum. For each interval δ in which the control state is constant, the sum uses the control information $k|_\delta$ to demultiplex the input $(k,a)|_\delta$ to the appropriate node (or *activity*, in the HyChart context) and to multiplex the output $b|_\delta$ to $(k,b)|_\delta$. Section 3.5.2 will show how the analog part Ana is constructed from the activities in a HySChart by using the $+$ operator.

As we demand that every primitive node be time guarded, the hierarchic node defined by a time-extended additive hierarchic graph consisting of the above operators also is time guarded. A time-extended additive feedback operator is not needed in the context of HyCharts. Therefore, no definition is given for it.

The Connectors: For HyCharts we only need the time-extended additive identity i_A. It is defined by

$$\mathsf{i}_A \;\subseteq (\mathcal{I} \times A^{\mathcal{T}}) \times A^{\mathcal{T}}, \qquad \mathsf{i}_A \;= \{((x,a),a) \mid a \in A^{\mathcal{T}} \;\wedge\; x \in \mathcal{I}\}$$

Hence, the state is not affected by the identity.

3.3.5 Refinement in the Multiplicative and Additive Models

In this section we introduce a notion of refinement and relate it to the above models for hierarchic graphs. The compositionality results described here are a mathematical basis for the refinement methods considered in Chapters 4 and 5.

We say that relation A is a refinement of relation B, written as $A \preceq B$, iff $A \subseteq B$. Monotony of the additive and multiplicative graph operators w.r.t. set inclusion ensures that refinement is compositional, i.e. in the additive as well as in the multiplicative model $A \preceq B$ implies $(A\,;C) \preceq (B\,;C)$, $(A \star C) \preceq (B \star C)$ and $(A\uparrow) \preceq (B\uparrow)$

holds, where in each case the type of relation C is such that it can be composed with A. Symmetrically, $(C\,;A) \preceq (C\,;B)$ and $(C \star A) \preceq (C \star B)$ holds, if $A \preceq B$. Apart from feedback, which is not defined in the time-extended additive model, the same results hold for the time-extended additive model.

3.4 Architecture Specification – HyACharts

The system architecture specification determines the interconnection of a system's components.

Graphical syntax. The architecture specification is a hierarchic graph, a so-called HyAChart (Hybrid Architecture Chart), whose nodes are labeled with component names and whose arcs are labeled with channel names. Each node may have subnodes. The node names and channel names only serve for reference. We use a graphical representation that is analogous to the structure specifications in ROOM [SGW94].

Semantics. As a HyAChart is a hierarchic graph, it is constructed with the operators of Section 3.3.1. Writing the graph as the equivalent relational expression and using the multiplicative model to interpret the operators in it directly results in the HyAChart's semantics.

As $\star$ is interpreted as the product operation for sets in this model, visual attachment here corresponds to parallel composition. Hence, each node in the graph is a component acting in parallel with the other components and each arc in the graph is a channel describing the data-flow from the source component to the destination component, as explained in Section 3.3.2.

Characteristics of components and channels. The component names in the graph refer to input/output behaviors specified in other HyACharts, in HySCharts (Section 3.5) or with other formalisms (Section 3.6). The channel names are the input and output variable names used in the specification of the components. We therefore do not distinguish between a channel and the variable carrying the channel's value at each time instant. The channels' types must be specified separately.

We distinguish three basic kinds of channels w.r.t. their time varying character and require that every channel in a HyAChart belongs to one of these kinds. The considered kinds are *discrete channels*, *continuous channels*, and *hybrid channels*. The value on a discrete channel only changes discretely at isolated, not necessarily equidistant time instants. Between those instants it is constant. The value on a continuous channel *only* changes continuously over time. For a hybrid channel, discrete changes in the channel's value may occur at isolated time instants and between the instants the value must evolve continuously. Infinitely many discrete changes during a finite time period are not allowed in all kinds of channels. By convention, all of these kinds are encoded as tuples of dense streams, i.e. as tuples of piecewise smooth, piecewise

Lipschitz continuous functions.[13] Every discrete or hybrid channel c with values of type D is encoded as a tuple of dense streams $(c.val, c.t) \in (D \times \mathbb{R}_+)^{\mathbb{R}_{p+}}$, where $c.val$ always holds the current value on the channel and $c.t$ holds a *time stamp* which at any time instant denotes the most recent time when a discrete change was made on the channel. The sender of the discrete or hybrid stream is responsible for specifying correct time stamps.

A continuous channel is encoded as a single dense stream without further components for modification times. As we will see, the stream of time stamps allows a HySChart to react with delay to a discrete change in its input. For discrete channels that only transmit events, the stream of values is not needed since the presence of an event can be encoded as a change in the stream of time stamps (see also Section 3.5). Although the range of time stamps is $\mathbb{R}_+$, we do *not* use the natural topology on $\mathbb{R}$ for it, but instead use the discrete topology (see Appendix B.2 for definitions of these topologies). With the streams of time stamps being piecewise constant, and hence piecewise continuous w.r.t. the discrete topology, this will permit more flexible predicates over the time stamps in HySCharts in the next section. For instance, a predicate like $x = y$ for time stamps x and y is an *open* set w.r.t. the discrete topology. This is sometimes needed in HySCharts.

From a methodological point of view, the type of channels should not only restrict the type of the values transmitted on them, but it should also specify the characteristics of its evolution over time. For discrete and hybrid channels, the type should therefore specify the *minimum event separation*, i.e. the minimal time between two discrete changes on the channel. For hybrid and continuous channels, it should specify the Lipschitz constant or bandwith with which the channel's value varies during any period of continuous evolution. For a stream $\sigma \in (D \times \mathbb{R}_+)^{\mathbb{R}_{p+}}$ flowing on a discrete or hybrid channel, the minimum event separation $mes(\sigma)$ is defined based on the time stamp information:

$$mes(\sigma) = inf\{x' - x \mid x, x' \in \mathbb{R}_+ \ \wedge \ x < x' \ \wedge \ \sigma.t(x) \neq \sigma.t(x')\}$$

For a stream τ flowing on a hybrid or continuous channel with values of type D, the Lipschitz constant $Li(\tau)$ constraining its continuous periods of evolution is defined as:

$$Li(\tau) = sup\{l \mid \forall \delta \in Int.\, cont(\tau.val|_\delta) \Rightarrow \forall x, y \in \delta.\, d(\tau.val(x), \tau.val(y)) \leq l \cdot |x - y|\}$$

where Int denotes the set of all left closed and right open intervals over $\mathbb{R}_+$, predicate $cont(x)$ is true iff x is a continuous function and d is the metric used on D. A bandwidth for the continuous periods of evolution of τ can be defined similarly. Note that the minimum event separation can be 0 or the Lipschitz constant can be ∞, respectively, for a dense stream. Furthermore, note that the Lipschitz constant of a signal and its bandwidth are related. For instance, a periodic signal $c(t) = a \cdot \sin(2\pi f \cdot t)$

[13] Note that smoothness and Lipschitz continuity are also defined for domains different from the real numbers in Section 3.2.3.

with frequency f is Lipschitz continuous with constant $a \cdot 2\pi f$. This follows from taking the maximum of its derivative.

Specifying constraints like a minimum event separation or a Lipschitz constant for a channel amounts to introducing assumptions and commitments for the components using these channels. For the sender on a channel, the constants are commitments which its output streams must satisfy, and for the receiver they are assumptions about its input streams. The refinement technique in Chapter 5 requires minimum event separations that are greater than 0 and finite Lipschitz constants. There, examples for such evolution constraints are also given.

We can now return to the HyAChart of our example system given in the introduction in Figure 3.2, left, and develop its semantics.

Example 3.5 (HyAChart of the EHC.) In Figure 3.2, left, the discrete channel *bend* transmits events which signal to the controller whether the car is in a curve. The continuous, real-valued channel *sHeight* transmits the chassis level measured by the sensors. The hybrid, real-valued channel *fHeight* forwards the filtered chassis level. The continuous real-valued channel *aHeight* transmits the effect of the actuators (compressor and the escape valve) on the chassis level. The discrete channels *reset* and *dReset* (delayed reset) transfer the reset event to the filter. Like channel *bend*, these channels forward streams of time stamps. The delay component D ensures that the feedback is well defined (see Section 3.3.2). At this point we do not fix event separations and Lipschitz constants for the channels.

The type of the filter, the control component and the delay component follow from the channels' types:

$$\begin{array}{lcl} \mathit{Filter} & \in & \mathbb{R}^{\mathbb{R}_{p+}} \times {\mathbb{R}_+}^{\mathbb{R}_{p+}} \to \mathcal{P}((\mathbb{R} \times \mathbb{R}_+)^{\mathbb{R}_{p+}}) \\ \mathit{Control} & \in & {\mathbb{R}_+}^{\mathbb{R}_{p+}} \times (\mathbb{R} \times \mathbb{R}_+)^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathbb{R}^{\mathbb{R}_{p+}} \times {\mathbb{R}_+}^{\mathbb{R}_{p+}}) \\ D & \in & {\mathbb{R}_+}^{\mathbb{R}_{p+}} \to \mathcal{P}({\mathbb{R}_+}^{\mathbb{R}_{p+}}) \end{array}$$

The semantics of the whole system EHC is defined as below. It is the relational algebra term corresponding to the HyAChart of Figure 3.2, left.

$$\begin{array}{lcl} \mathit{EHC} & \in & {\mathbb{R}_+}^{\mathbb{R}_{p+}} \times \mathbb{R}^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathbb{R}^{\mathbb{R}_{p+}}) \\ \mathit{EHC} & = & ((\mathsf{I} \times \mathit{Filter}) \,;_\times\, \mathit{Control} \,;_\times\, (\mathsf{I} \times D)) \uparrow^{\mathbb{R}}_{\times} \end{array}$$

Note that in order to specify a system architecture, the user only has to draw the HyAChart and to define the types of the channels, including event separations and Lipschitz constants, if desired. A sufficient condition for the well definedness of the system architecture is that there is a non-zero delay on every feedback channel. □

3.5 Component Specification – HySCharts

A HySChart (Hybrid StateChart) defines the discrete and the analog part of a hybrid machine. The input/output behavior of the resulting component follows from these

parts as explained in Section 3.2.

Graphical syntax of HySCharts. A HySChart is a hierarchic graph, where each node is of the form depicted in Figure 3.15, left. Each node may have subnodes. It is labeled with a node name, an activity name and possibly the symbols $\rightarrow\!\!\circ$ and $\circ\!\!\rightarrow$ to indicate the existence of an entry or exit action, which is executed when the node is entered or left. The outgoing edges of a node are labeled with action names. The action names stand for predicates concerning the input, the latched data state and the next data state. They consist of a *guard* and a *body*. The node name refers to the node's state invariant, which is a predicate of the same type as actions. The activity names refer to systems of ordinary differential equations or differential constraints. The specification of actions and activities and their semantics is explained in detail in the following sections. Transitions from composed nodes express preemption. If a transition traverses hierarchic levels in a HySChart, it is split into segments at the boundary of every hierarchic node, and an entry or exit point symbol is drawn there, depending on whether the transition is incoming or outgoing for the respective node. The entry point symbol is $\otimes$ and the exit point symbol is $\odot$. The symbol is juxtaposed with a label referring to the transition at the upper hierarchic level. In Figure 3.3 an example for a HySChart is depicted. There, the hierarchic levels in the chart have been put besides each other (see also Section 3.1). Except for activities and invariants, HySCharts look similar to ROOMcharts [SGW94].

Semantics of HySCharts. The semantics of a HySChart is divided into a discrete and an analog part. The discrete part follows via syntactic transformations from the diagram. The analog part is constructed from the chart with little effort.

In the following sections we will first explain how the discrete part is derived from a HySChart, and then we cover the analog part.

3.5.1 The Discrete Part

A HySChart is a hierarchic graph and therefore constructed from the operators in Section 3.3.1. As mentioned in Section 3.3.3, interpreting the graph in the additive model leads to an analogy to automata diagrams. This is utilized in the semantics definition for the discrete part. The basic idea is to regard the nodes in the HySChart as macros for certain additive hierarchic graphs and to construct a state-transition relation, i.e. the discrete part Com, from the HySChart by expanding these macros. There are two important points to note. First, in the resulting additive hierarchic graph no time passes in any node or subgraph. Computation is instantaneous. Second, the macro expansion introduces so called wait entry and wait exit points in the resulting additive hierarchic graph. These points are not drawn by the designer, but they are implicit in the HySChart. On one of the wait exit points, the graph outputs the next data state, which it computes from the current input and the latched state.

This state is received by the analog part of the hybrid machine model and then, via Lim, the graph gets a latched data state back on one of its wait entry points.

By construction of the additive hierarchic graph, every wait entry/exit point uniquely identifies a primitive node of the HySChart (and also all hierarchic nodes in which the primitive node is contained). As usual in automata diagrams, we also call the nodes in HySChart *control states*. Thus, when the additive hierarchic graph outputs a data state on a certain wait exit point, this in other words means that the state-transition relation has selected the corresponding node in the HySChart as the currently active control state.

If the additive hierarchic graph outputs the received data state without modification along the wait exit point which corresponds to the same control state as the entry point on which it received the data state, this means that the discrete part idles. On the syntactic level, i.e. on the level of the HySChart, we also say that *control resides* (or *is*) in the respective control state of the HySChart.

In this section we first define the (sub)graphs by which the nodes in the HySChart are replaced to obtain the additive hierarchic graph that defines the discrete part. These (sub)graphs are also called *computation units*, because they can modify the data and control state. After that, the actions and invariants of HySCharts are elaborated.

3.5.1.1 Computation Units

Each primitive node of the HySChart represents the graph given in Figure 3.15, top right. The graph formally corresponds to the relational expression below:

$$CompUnit \;=\; (+_{i=1}^{m} entry \;+\; \mathsf{i}_{\mathcal{S}})\,;_{+\ m+1}\!\!\succ\!\!\bullet_{\mathcal{S}}\,;_{+}\ \bullet_{\mathcal{S}}\!\!\prec_{n+1}\,;_{+}\ ((+_{i=1}^{n} action_i\,;_{+}\ exit) + Inv)$$

where $\mathcal{S}$ is the data state space of the HySChart, as before. According to the additive operators, the graph has the following intuitive meaning: A computation unit receives the control at one of its entry points en_i and gives the control back at one of its exit points ex_j.

After receiving control at a regular entry point, i.e. an entry point different from the wait entry wt, a computation unit executes its entry action *entry*, if one is specified. Then it evaluates a set of *action guards*.[14] If one of the guards is true, then the corresponding *action* is said to be *enabled* and its *body* is executed. After finishing its execution, the computation unit executes its exit action *exit*, if present. Finally, control is given to another computation unit along the exit point corresponding to the executed action. Invariant *Inv* is treated similarly to action guards. However, no body and exit action exists for it.

If more than one guard and/or the invariant is true, then the computation unit *non-deterministically* chooses one of them. If the invariant is chosen, then the discrete

[14] An action $action_k$ consists of a *guard* and a *body*.

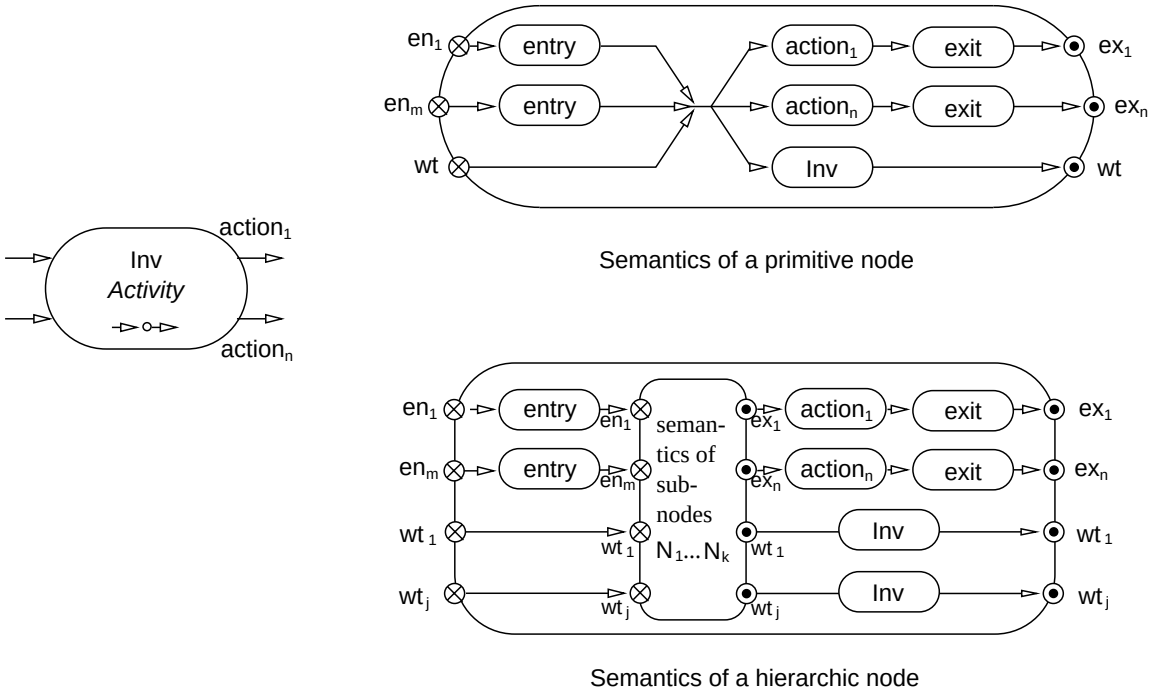


Figure 3.15: Syntax (left) and semantics (right) of a node in a HySChart.

computation is completed, and the discrete part outputs the computed data state along the designated wait exit point *wt*. Section 3.5.2 shows that the analog part takes advantage of the information concerning the exit point to determine which activity is to be executed and gives (via *Lim*) a data state back along the corresponding wait entry point. This can also be regarded as *busy waiting* in the computation unit.

Note that the kind of composition of action guards and invariant (visual attachment after ramification) depicted in Figure 3.15 provides that the invariant need not be true at the moment when the node is left via an action. This differs from the semantics of the popular hybrid automata formalism [ACH+95], where the invariant is required to hold true when a state is left via one of its outgoing transitions. In our view the choice made for HySCharts is rational, since when specifying with hybrid automata designers often choose invariants as negations of action guards, such that at any point in time only one of them can be true.

In order to avoid *time deadlock*, i.e. situations in which neither one of the action guards nor the invariant is true, invariants must be chosen with care. On the level of semantics, time deadlock leads to components which are not total in their input, but yield an empty set of streams for some inputs. (Note that the type of components does not permit finite output streams.) Thus for primitive nodes we require that at least one action guard or the invariant is true for any input and latched data state.

3.5.1.2 Hierarchy

A composed or *hierarchic* node in the HySChart whose outgoing transitions all stem from one of its subnodes represents the additive graph in Figure 3.15, bottom right.[15] To avoid confusion with hierarchic nodes in the HySChart we call such a graph a *hierarchic computation unit*. It contains the semantics, i.e. the computation units, of all the hierarchic node's subnodes in the HySChart. A principal difference from (primitive) computation units is that the entry points of the hierarchic computation unit are not identified; instead they are connected to the corresponding entry points of the contained sub-computation units. Similarly, the exit points of the subunits are connected to the corresponding exit points of their enclosing hierarchic computation unit. In addition, the hierarchic unit has a wait entry and wait exit point for every wait entry/exit point of the subunits. When it receives the data state on one of them, the data state is directly passed on to the wait entry point of the corresponding sub-computation unit. Thus, the wait entry point identifies a subunit. Under the assumption that the hierarchic node's invariant *Inv* holds, the hierarchic computation unit is left along a wait exit point, if a subunit is left along its corresponding wait exit point. Note that the invariant of the hierarchic node in the HySChart is duplicated in the hierarchic computation unit for every wait exit point of a subunit. This is needed, because the exit point information of the subunit must not be lost when the invariant is checked. To ensure that the hierarchic invariant indeed is true when a subunit is left along a wait exit point, we require that the invariant is implied by the disjunction of the invariants of its enclosed computation units. In presence of preemption this requirement is relaxed (see Section 3.5.1.5). Enforcement of this condition is motivated by the desire to regard the hierarchic levels in a HySChart in isolation.

3.5.1.3 Actions and Invariants

An *action* a is a relation between the current input, the latched data state and the next data state:

$$a \subseteq (\mathcal{I} \times \mathcal{S}) \times \mathcal{S}$$

For HySCharts, actions are specified by their characteristic predicate. Such a predicate is the conjunction of a precondition (the *action guard*) on the latched data state and the current input and a postcondition (the *action body*) that determines the next data state. The precondition implies that the postcondition is satisfiable, hence the action is enabled iff the precondition is true. Invariants are of the same type as actions and are similar to action guards. In particular, they do not modify the data state. In the characteristic predicates we use *left-quoted variables* v‵ to denote the current input, *right-quoted variables* v' to denote the next data state and *plain variables* to

[15] Hierarchic nodes with outgoing transitions that do *not* originate from one of its subnodes are covered in the following paragraph on preemption.

denote the latched data state. Moreover, we mention only those variables which are modified in the action explicitly. For the others (also in invariants), we always assume the necessary equalities which state that they do not change. To simplify notation further, we associate a variable c with each channel c. For variables associated with discrete or hybrid channels, we write $c.val^{\backprime}$ for the channel's current value and $c.t^{\backprime}$ for the current time stamp. As indicated in Section 3.2.1, for each time stamp associated with a discrete or hybrid input channel c there is a private copy in a component's data-state space, the *latched time stamp*, denoted by $c.t$. It is used by the component to remember the time stamp of the last input it processed. For example, the action of resetting the filter in the EHC example is defined as follows:

$$dReset.t^{\backprime} \neq dReset.t \wedge dReset.t' = dReset.t^{\backprime} \wedge fHeight.val' = 0 \wedge fHeight.t' = now$$

The first conjunct in the predicate is the action guard, the rest is the action body. The guard is true if *dReset* changes. When this is the case, the latched copy of *dReset* is updated (2nd conjunct) and *fHeight* is reset to 0 (3rd conjunct). Variable *now* is the current time in the EHC. Whenever variables that are associated with discrete or hybrid output channels are modified in an action body, the respective time stamp for the channel must be set to *now* (4th conjunct). In a tool environment, this convention can automatically be enforced by the tool. In Section 5.2.1, we elaborate further on the syntax for action guards. Note that if the action guard is false, the ramification in a computation unit will nondeterministically select another action (or the invariant).

In conjunction with hierarchy, which can lead to the sequential composition of actions, the action guards must be chosen with care in order to guarantee that the discrete part specified by the HySChart is total. We require that the first action in a sequence be chosen such that all succeeding actions are guaranteed to be enabled. If ramification is involved in the sequential composition of actions, it is sufficient when one of the actions in each ramification is guaranteed to be enabled. Similarly, tools for Statechart-like notations often enforce that no guard can be specified for the succeeding transition segments.

Events. Latched time stamps allow us to model many different communication styles. Particularly interesting for our example are *events* which we model by changing the modification time stamp associated with a discrete channel e. The occurrence of an event is detected by testing if the current time stamp for that channel is different from the latched time stamp, i.e. $e.t^{\backprime} \neq e.t$. We write $e?$ as abbreviation for $e.t^{\backprime} \neq e.t \wedge e.t' = e.t^{\backprime}$. (The second part of the conjunction updates the latched time stamp of e.) Similarly, the sending of an event is specified by $e.t' = now$ which is abbreviated by $e!$, where *now* is the component's private clock. With this notation, the filter reset action can be rewritten as $dReset? \wedge fHeight' = 0$. *Message passing* over a discrete channel c is modeled in the same way. This time, however, the discrete channel not only transmits a time stamp $c.t$ but also a value $c.val$, namely the message. Checking the arrival of message a on channel c is denoted by the following expression: $c? \wedge c.val^{\backprime} = a$.

We abbreviate this by $c?a$. Similarly, sending the message a on the channel c is given by $e! \wedge c.val' = a$ which is abbreviated by $c!a$.

As in the case of events and messages, the current and latched time stamps that are associated with hybrid channels can be used to detect discrete jumps in their value.

3.5.1.4 Preemption

In Statecharts a *preemptive* (or group) transition is a transition that may be taken from all substates of a hierarchic state. In HySCharts we use transitions originating directly from a hierarchic node (and not from any of its subnodes) to express preemption. The actions associated with such transitions are called *preemptive actions*. As discussed in [GSB98b], one can define such a preemptive action as having higher priority than any action inside the hierarchic node (strong preemption) or as having lower priority than any action inside the node (weak preemption). Here we use weak preemption because it is simpler and it allows actions inside a hierarchic node to overwrite the preemptive action.

The corresponding computation unit (or additive graph) for a hierarchic node with preemption is obtained as follows. First, we replace the hierarchic node with preemptive actions $pa_1, \ldots, pa_h$ (Figure 3.16, top left) by its corresponding graph of Figure 3.15, bottom right, but *without* the hierarchic node's invariant. It is treated in a second step. This first step yields a graph of the form given in Figure 3.16, top right.[16] In a second step, we obtain the hierarchic computation unit for the original node with the preemptive transitions, by replacing the graph in Figure 3.16, top right, by the graph in Figure 3.16, bottom. The subnode in the left part of the graph is exactly the graph in the node in Figure 3.16, top right.

This hierarchic computation unit basically expresses that whenever a sub-computation unit is left on a wait exit point *wt* and one of the preemptive actions is enabled, it can be executed and is followed by the exit action *exit*, which is associated with the hierarchic node in the HySChart. The hierarchic computation unit is then left along an exit point *pex* corresponding to the executed preemptive action. A preemptive action *must* be executed if no other preemptive action is enabled and the hierarchic node's invariant *Inv* is false.[17] If the invariant is true and selected nondeterministically (by ramification), the hierarchic computation unit is left along the wait exit point that corresponds to *wt* of the subunit, i.e. the subunit information is maintained.

In our example of Figure 3.3, left, the action *o2b* is a preemptive action of the composed node *outBend*.

[16] The interior of the graph is only depicted schematically here. It exactly is the graph of Figure 3.15, bottom right, but without the hierarchic node's invariant which would usually have to be evaluated on the way to a wait exit point of a hierarchic computation unit.

[17] As before, the hierarchic node's invariant is duplicated in the hierarchic compution unit for every wait exit point of a sub-computation unit.

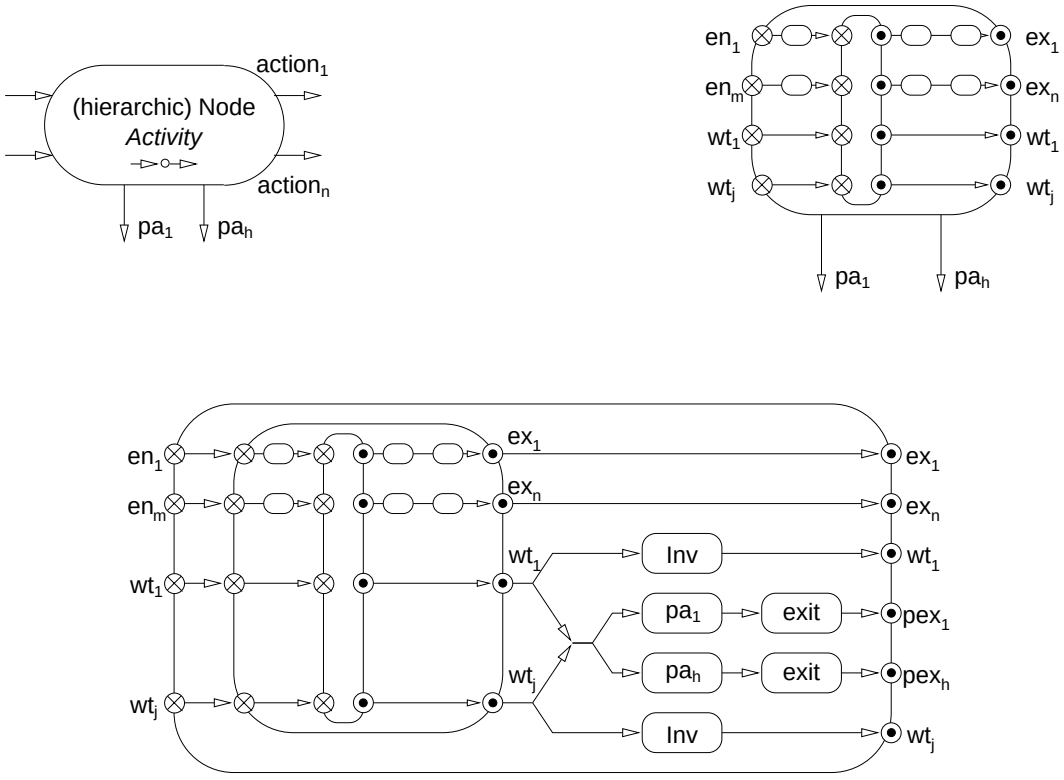


Figure 3.16: The semantics of preemption.

3.5.1.5 Sound Invariants

Hierarchy and invariants. Due to the syntactic transformation of HySCharts to additive hierarchic graphs, the nodes representing the invariant in a hierarchic computation unit can only be reached when coming from the wait exit points of the computation unit's subunits. These wait exit points on their part can only be reached via the respective subunit's invariant (see the arcs leading to boxes labeled *Inv* in Figure 3.15, bottom right, and Figure 3.16, bottom). To ensure that the hierarchic computation unit can be left in such cases, we demand that if the invariant of one of its subunits is true, either the hierarchic units's invariant also is true, or one of its preemptive actions, is enabled. Formally, we require that there exists a $(i, s', s'') \in Inv \cup \bigcup_{i=1}^{n} Guard_i$ for all $(i, s, s') \in \bigcup_{j=1}^{k} Inv_j$, where the $Guard_i$ refer to the action guards of the hierarchic computation unit's preemptive actions, Inv refers to its invariant and the Inv_j refer to the invariants of its subunits. For every hierarchic computation unit we may only consider preemptive actions, because exit from the unit on the other exit points is determined by the actions associated with the subunits. Hence, we demand that if the invariant of a subunit is true, either the invariant of its enclosing hierarchic computation unit also is true or one of its preemptive actions is enabled. As explained above, for a hierarchic computation unit without preemptive

transitions this means that the invariant must be implied by the disjunction of the invariants of its subunits. In terms of the HySChart's nodes, this requirement similarly means that if the invariant of a subnode is true, either the enclosing hierarchic node's invariant also is true or a transition emerging directly from the hierarchic node is enabled. This requirement enables us to regard the hierarchic levels in a HySChart in some isolation. Namely, leaving a sub-computation unit on a wait exit point cannot be prevented at higher hierarchic levels. In terms of the HySChart's nodes, this means that if no preemtive action of a hierarchic node is enabled, the node cannot prevent that control remains is a subnode whose invariant is true.

Furthermore, the above condition helps to avoid time deadlocks. Enforcing it, together with requiring that (1) every primitive computation unit can always be left via an action or the invariant and (2) all actions in a sequential composition are enabled if the first one is, results in a sufficient condition for totality of the discrete part that the HySChart defines.

Idling and invariants. In Section 3.2 we furthermore demanded that the discrete part may only idle on a topologically open subset of $\mathcal{I} \times n \cdot \mathcal{S}$, in order to ensure well definedness of the hybrid machine model. This condition is satisfied by a HySChart defining the discrete part, if the invariant of every node in the chart identifies a topologically open set in $\mathcal{I} \times \mathcal{S}$. This can be explained as follows: The hierarchy in a HySCharts leads to sequentially composed invariants, which on the level of semantics corresponds to their intersection (because invariants leave the data state unchanged). As there is only a finite number of hierarchic levels in a HySChart, the composed invariant also identifies an open set.

As far as the choice of invariants is concerned, using the negation of the disjunction of the actions guards of the transitions directly emerging from a node (and from none of its subnodes) is a sensible choice for the time being. It provides that transitions are *eager*, i.e. they are taken as soon as their guard is true, and it is compatible with our requirements for totality from above, which related invariants on different hierarchic levels, and invariants and action guards on the same level. We call such an invariant an *exact invariant*. If the syntax of actions is sufficiently simple, such invariants can be automatically computed by a CASE tool. If the guards are topologically closed sets, this also guarantees that the invariant is open. Note that action guards of the form $e.t^{\backprime} \neq e.t$, which are used to detect discrete changes and events, identify a *closed* set on $\mathcal{I} \times \mathcal{S}$ since we use the discrete topology for time stamps. The same holds for propositions over discrete variables with domains different from the real numbers. Furthermore, non-strict inequations of arithmetic expressions over real-valued variables also result in closed sets. Hence, the negation of a (finite) disjunction of guards built from such atoms leads to legal invariants. The version of HyCharts without invariants in [GSB98a, GS00b] uses such action guards and is equivalent to the version presented here, if invariants are derived from action guards as proposed above. Section 5.2.1 will explain how invariants which relax the eagerness of transitions

can systematically be derived from action guards. The choice of invariants presented there results in *delayable* transitions. They can, but need not, be taken as long as their guard is true. However, note that the selection of invariants can also be based on other principles.

3.5.1.6 Semantics of the Discrete Part

If each node in the HySChart is replaced by the corresponding computation unit of Figure 3.15, right, and 3.16, bottom, we obtain an additive hierarchic graph whose primitive nodes merely are relations, which correspond to the actions and invariants. Writing the graph as the corresponding relational expression with the additive operators gives the denotational semantics of the HySChart's discrete part Com, i.e. the discrete part of a hybrid machine.

At the highest level of hierarchy, the hierarchic graph resulting from the HySChart has one wait entry/exit point pair for every primitive (or leaf) node in the chart. Thus, a wait entry/exit point uniquely identifies a primitive node of the HySChart (and also all hierarchic nodes in which the primitive node is contained). On one of these entry points Com receives the latched data state from Lim, and similarly on one of the exit points it gives the next data state to the analog part. On the semantic level, the type of this input (output) interface is the n-fold sum $n \cdot \mathcal{S}$, where there is exactly one summand in the sum for every entry (exit) point. This also means that n is the number of primitive nodes in the HySChart. Together with the external input, this interface type defined by the entry/exit points provides that the discrete part's type is $(\mathcal{I} \times n \cdot \mathcal{S}) \to \mathcal{P}(n \cdot \mathcal{S})$, as required in Section 3.2.2 (where totality of Com is assumed). The analog part uses the entry/exit point information encoded in the disjoint sum to select the right activity for every node in the HySChart (Section 3.5.2).

Note that when Com gives the next program state $(k', s') \in n \cdot \mathcal{S}$ for current input i and latched program state (k, s), then in terms of the additive graph this means that s' is given on wait exit point k'. In the HySChart from which the additive graph for the discrete part is derived, k' identifies a primitive node and also all hierarchic nodes which contain this node. By construction of the additive graph for the HySChart all invariants of these nodes must be true for i and s', if s' is given on wait exit point k'. When we now use (k', s') as input for Com, the wait entry points provide that k' identifies the same primitive and hierarchic nodes as before. If we again use external input i, the invariants of these nodes are still true for i and s'. Thus, $(k', s') \in Com(i, (k', s'))$ holds, i.e. Com can idle for i and (k', s'). This was used in the proof in Section 3.2.4.

Finally, note that it is reasonable to call the nodes in a HySChart *control states*, because the construction of the discrete part ensures that every primitive node in a HySChart indeed corresponds to a control state on the level of semantics. When the context is clear, we therefore use the term *control state* (or simply *state*) to refer to

a node of a HySChart. This terminology is carried over to the hierarchic nodes in the HySChart. Thus, they are also called *hierarchic states* and its subnodes are also called *substates*.

3.5.1.7 Example

To outline the utility of this approach for hybrid systems we now return to the HySChart for the controller given in the introduction.

Example 3.6 (The EHC's *Control* component.) We describe the states and transitions of the HySChart in Figure 3.3 in a top-down manner. The activities, written in italics in the figure, are explained in the next section. For easier reference we repeat the HySChart here (Figure 3.17) and collect the definitions of all actions, invariants and activities occuring in it in Table 3.1. The invariants we use in this example result

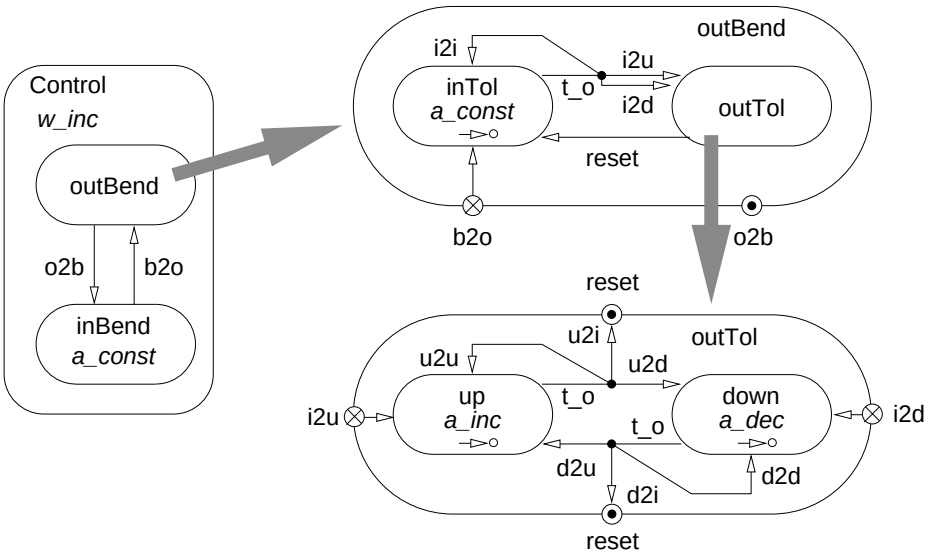


Figure 3.17: The EHC's *Control* component (repeated).

from the negation of the disjunction of the action guards of all transitions originating directly from a state (and from none of its substates). The example is selected in a way which allows us to present a large part of the various specification facilities in HySCharts. From a methodological point of view, a model like this should not occur in the development process if the refinement techniques of Chapter 5 are employed because the model mixes abstract specification fragments, like the continuous sensing of some input channels and vaguely specified activities, with implementation oriented fragments, like sampled reaction to other input channels.

The state *Control*. The top-level hierarchic state *Control* has two substates, *outBend* and *inBend*. When the controller senses that the car is in a curve, state

Actions:					
$b2o$	$\equiv$	$bend?$	$o2b$	$\equiv$	$b2o$
t_o	$\equiv$	$w \geq t_s$	$i2i$	$\equiv$	$lb \leq fHeight.val` \leq ub$
$i2u$	$\equiv$	$fHeight.val` \leq lb$	$i2d$	$\equiv$	$fHeight.val` \geq ub$
$reset$	$\equiv$	$reset!$	$u2u$	$\equiv$	$fHeight.val` \leq lb$
$u2i$	$\equiv$	$lb \leq fHeight.val` \leq ub$	$u2d$	$\equiv$	$fHeight.val` \geq ub$
$d2d$	$\equiv$	$fHeight.val` \geq ub$	$d2i$	$\equiv$	$lb \leq fHeight.val` \leq ub$
$d2u$	$\equiv$	$fHeight.val` \leq lb$	$entry$	$\equiv$	$w' = 0$
Invariants:					
$Control_{inv}$	$\equiv$	$true$	$inBend_{inv}$	$\equiv$	$\neg(bend?)$
$outBend_{inv}$	$\equiv$	$inBend_{inv}$	$inTol_{inv}$	$\equiv$	$w < t_s$
$outTol_{inv}$	$\equiv$	$true$	up_{inv}	$\equiv$	$w < t_s$
$down_{inv}$	$\equiv$	$w < t_s$			
Activities:					
w_inc	$\equiv$	$\dot{w} = 1$	a_const	$\equiv$	$\frac{d}{dt} aHeight = 0$
a_inc	$\equiv$	$\frac{d}{dt} aHeight \in [cp_-, cp_+]$	a_dec	$\equiv$	$\frac{d}{dt} aHeight \in [ev_-, ev_+]$

Table 3.1: Definition of actions, invariants and activities of the *Control* component.

inBend is entered. It is left again when the controller senses that the car is no longer in a curve. Sensing a curve is event-driven. We use the discrete channel *bend* with its associated time stamp for this purpose. The actions $o2b$ and $b2o$ are identical and very simple: $o2b \equiv b2o \equiv bend?$ The invariant of *Control* is *true*. States *inBend* and *outBend* have invariant $inBend_{inv} \equiv outBend_{inv} \equiv \neg(bend?)$.

The state *outBend*. State *outBend* is refined to *inTol* and *outTol* as shown in Figure 3.17, top right. Control is in *inTol* as long as the filtered chassis level is within a certain tolerance interval. The compressor and the escape valve are off. If *fHeight* is outside this interval at a sampling point, one of the substates of *outTol* is entered. These substates are left again, when *fHeight* is inside the desired tolerance again and the filter is reset then. The actions originating from *inTol* are defined as follows:

$$t\_o \equiv w \geq t_s, \qquad i2i \equiv lb \leq fHeight.val` \leq ub$$
$$i2u \equiv fHeight.val` \leq lb, \qquad i2d \equiv fHeight.val` \geq ub$$

Note that $fHeight.val`$ refers to the current value received on the hybrid input channel *fHeight* from the filter component. An interesting aspect of *inTol* is the specification of the composed action started by the timeout t_o, which semantically corresponds to the ramification operator for hierarchic graphs. Alternatively, one could use three separate transitions instead; however, in this case the visual representation would fail to highlight the common enabling condition t_o. The invariant of *inTol* is $w < t_s$, i.e. the negation of t_o. It does not reflect the informal meaning of *inTol*, being the state in which *fHeight* is inside the tolerance interval, because the outgoing transition of *inTol* is solely triggered by the timeout.

Leaving the state *outTol* along transition *reset* causes the execution of the *reset* action. This action is always enabled and defined by $reset \equiv reset!$. Note that we use the same name for the action and its associated event here. The invariant of *outTol* is true, because there is no transition emerging directly from it. (The *reset* transition does not originate directly from it, but from its substates.)

The transition *o2b* originates from the hierarchic state *outBend* (and from none of its substates). This expresses weak preemption, i.e. this transition can be taken from any substate of *outBend*, as long as it is not overwritten at lower hierarchic levels.

The state *outTol*. As shown in Figure 3.17, bottom right, the state *outTol* consists of the substates *up* and *down*. When the filtered chassis level is too low at a sampling point, state *up* is entered, where the compressor of the EHC system is on. When the level is too high, *down* is entered, where the escape valve of the system is open. Control remains in these states until *fHeight* is inside the desired tolerance again (actions $u2i, d2i$). These actions cause *outTol* to be left along transition *reset*, indicated by label *reset* of the exit points. The actions originating from *up* and *down* are very similar to those of *inTol*:

$$u2u \equiv fHeight.val` \leq lb, \quad u2i \equiv lb \leq fHeight.val` \leq ub, \quad u2d \equiv fHeight.val` \geq ub,$$
$$d2d \equiv fHeight.val` \geq ub, \quad d2i \equiv lb \leq fHeight.val` \leq ub, \quad d2u \equiv fHeight.val` \leq lb$$

Again, ramification is used in the chart to highlight the common enabling condition t_o for these actions. Similar to *inTol*, the invariant of *up* and *down* is $w < t_s$ because the transitions originating from these states are triggered only by the timeout.

As indicated by the symbol $\rightarrow\!\circ$, the states *inTol*, *up* and *down* have an entry action. It is defined as $entry \equiv w' = 0$ and resets w. Together with action t_o and the activity w_inc, it models sampling in these states, i.e. all transitions directly originating from these states can only be taken at the end of a sampling interval.

Semantics. The discrete part follows directly from the HySChart by replacing the nodes in the HySChart by their corresponding computation units of Figure 3.15, right, and 3.16, right. As an example, Figure 3.18 depicts the resulting additive hierarchic

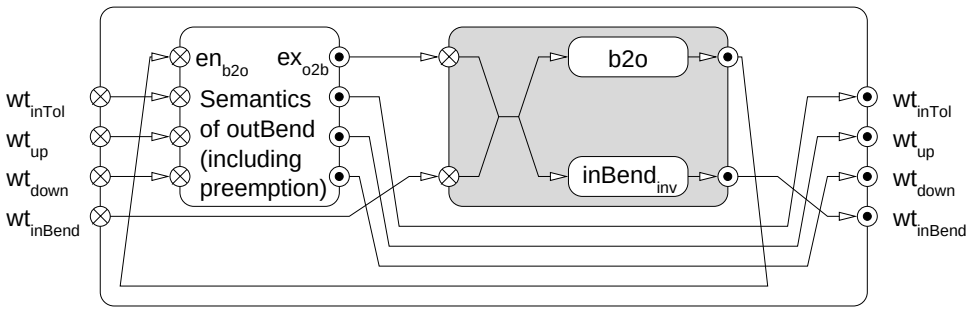


Figure 3.18: Additive hierarchic graph for *Control*.

graph for node (or state) *Control*. It is derived from the HySChart in Figure 3.17. The (sub)computation unit for state *outBend* which also covers the preemptive transition *o2b* is not depicted in the figure. Its structure is similar to the graph in Figure 3.16, bottom. The (sub)computation unit for state *inBend* is the subgraph highlighted in gray in the figure. The corresponding textual notation is:

$$\begin{aligned} \mathit{Control} &= ((\tbinom{1}{4} ;_+ \mathit{outBend} + \mathsf{i} ;_+ \mathsf{i} + \tbinom{1}{3} ;_+ \mathit{inBend} + \mathsf{i} + \mathsf{i} + \mathsf{i} ;_+ \tbinom{4}{1}) \uparrow^1_+) ;_+ \tbinom{3}{1} \\ \mathit{inBend} &= {}_2\!\!\succ\!\!\bullet ;_+ \bullet\!\!\prec_2 ;_+ (b2o + \mathit{inBend}_{inv}) \end{aligned}$$

where *b2o* and inBend_{inv} are defined as above and the definition of *outBend* is not unfolded.

Every wait entry/exit point pair at the highest hierarchic level of the additive graph which is derived from a HySChart corresponds to a summand in the n-fold sum in the type of the HySChart's discrete part. For *Control*, this means that its discrete part has type:

$$\mathit{Com} \in (\mathcal{I} \times 4 \cdot \mathcal{S}) \to \mathcal{P}(4 \cdot \mathcal{S})$$

This reflects the four primitive nodes (or states) *inBend*, *inTol*, *up* and *down* in the HySChart. Here, the input space and the data-state space are defined by $\mathcal{I} = \mathbb{R}_+ \times (\mathbb{R} \times \mathbb{R}_+)$, which corresponds to discrete channel *bend* and hybrid channel *fHeight*, and $\mathcal{S} = \mathbb{R} \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+$, which corresponds to continuous output channel *aHeight*, discrete output channel *reset*, latched time stamps for channels *bend* and *fHeight* and further private variables w and *now*. □

Note that the user only has to draw the HySChart and give the definitions of the actions and invariants, or specify how to derive invariants from actions. The corresponding discrete part can be constructed automatically.

3.5.2 The Analog Part

The second part of a HySChart's semantics is the analog part it defines. In this section we explain how this analog part is derived from the chart.

Syntax of activities. Each activity name in the HySChart refers to a system of ordinary differential equations and differential constraints on the variables of the component and the input. To simplify notation of the equations, we implicitly assume suffix *.val* for the variables in an activity definition which are associated with hybrid channels. Time stamps and variables associated with discrete channels are not changed by activities. The concrete syntax of activities a is as follows:

$$a ::= \dot{\vec{x}} \# f(\vec{x}, \vec{i}) \mid a \wedge a$$

where $\# \in \{\geq, \leq, =, >, <, \neq\}$, $\vec{x}$ is a tuple of real-valued controlled variables, excluding time stamps, and $\vec{i}$ denotes the tuple of all real-valued current inputs, excluding

time stamps. f is a function over $\vec{x}$ and $\vec{i}$. Usually tuple $\vec{x}$ only contains *some* real-valued controlled variables. This reflects that the activity only constrains the evolution of some, but not all, controlled variables. Interval constraints like $\dot{x} \in [c_1, c_2]$ can be reduced to the above syntax and are regarded as macros. Note that, in principle, continuous time block diagrams (Section 2.1.2) may also be used for activity specification, as standard methods exist for transforming them to differential equations provided they only contain blocks for the arithmetic operations, integration and differentiation.

Example 3.7 (The activities of *Control*.) In our example in Figure 3.17, the activity names written in italics stand for the following differential constraints:

$$\begin{array}{lcllcl} w\_inc & \equiv & \dot{w} = 1 & a\_inc & \equiv & \frac{d}{dt} aHeight \in [cp_-, cp_+] \\ a\_const & \equiv & \frac{d}{dt} aHeight = 0 & a\_dec & \equiv & \frac{d}{dt} aHeight \in [ev_-, ev_+] \end{array}$$

where $cp_-, cp_+ > 0$ and $ev_-, ev_+ < 0$ are constants. The notation $\dot{x}$ and $\frac{d}{dt}x$ is used interchangeably to denote the time derivative of x. For short variable names the "dot"-notation is more convenient. Due to activity w_inc, w evolves in pace with physical time. Variable *aHeight*, which models the influence of the controller on the chassis level (Figure 3.2), either increases at a rate that is in the interval $[cp_-, cp_+]$ (activity a_inc), decreases (a_dec) or remains constant (a_const).

This is all the user has to provide to specify the analog part. □

Semantics of activities. The semantics $[\![a]\!] \subseteq (\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}}) \times \mathcal{S}^{\mathbb{R}_{s+}}$ of an activity a is defined in the following way:

$$\begin{array}{lcl} [\![\dot{\vec{x}} \# f(\vec{x}, \vec{i})]\!] & = & \{((\iota, \sigma), \sigma) \in (\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}}) \times \mathcal{S}^{\mathbb{R}_{s+}} \mid \\ & & \quad \forall t \geq 0 . \dot{\sigma}.\vec{x}(t) \# f(\sigma.\vec{x}(t), \iota.\vec{i}(t))\} \\ [\![a_1 \ \wedge \ a_2]\!] & = & [\![a_1]\!] \cap [\![a_2]\!] \end{array}$$

where $\# \in \{\geq, \leq, =, >, <, \neq\}$ and for an input or state stream χ we write $\chi.\vec{y}$ to denote those components of χ which correspond to the variable names in tuple $\vec{y}$. Similarly $\dot{\chi}.\vec{y}$ denotes the tuple of the derivatives of those components of χ which correspond to the variable names in $\vec{y}$. As we only consider smooth functions in activities and as the variables in tuple $\vec{x}$ are required to be real-valued, the derivative is guaranteed to exist. All controlled variables which are not a component of $\vec{x}$ as well as the components of $\vec{x}$ at time 0 are not constrained by the activity $\dot{\vec{x}} \# f(\vec{x}, \vec{i})$. At the start of a smooth segment, the start value of $\vec{x}$ is determined by *Com* and *Lim* of the machine model. Note that the received state-stream and the output state-stream of the activity coincide. The semantics definition for conjunction is as expected. Obviously, conjunction preserves that the received state-stream and the output state-stream of an activity coincide. Hence, for all activities a, $((\iota, \sigma), \tau) \in [\![a]\!]$ implies $\sigma = \tau$. If no activity is specified for a node in a HySChart, the time-extended additive identity

relation ¡ restricted to smooth streams is used as activity semantics for that node. Thus, no further constraints on the received state-stream are added this way.

By definition, activities are not total in $\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}}$, but only allow inputs ι, σ for which the respective constraints are satisfied. In this case, their output is the same state-stream σ they received. In interaction with the other parts of the machine model, in particular the feedback loop and *Com*, this property of activities provides that *Com* and the activities must agree on the same smooth evolution of the component's state such that *Com* can idle for this evolution. As we will see below, usually *resolvability* of the differential constraints used as activities is required. This implies that the constaints must be satisfiable for any smooth external input stream $\iota \in \mathcal{I}^{\mathbb{R}_{s+}}$.

Note that the above general syntax includes constraints of the form $\dot{x} = 0$ denoting that real-valued variable x remains constant. For discrete variables and time stamps, the restriction of activities to smooth streams guarantees that they are not modified by activities. The reason for this is that we use the discrete topology for these variables. In this topology the evolution of a variable is continuous iff it is constant. This concludes the syntax and semantics definition of activities. In the following we argue on the level of activity semantics and use the term activity to refer to semantics, not syntax.

Hierarchic composition of activities. To reflect the hierarchy in the HySChart the semantics of the activities specified in the nodes are composed appropriately with the time-extended additive operators introduced in Section 3.3.4. We use sequential composition to compose activities at successive levels of hierarchy and the (time-extended) disjoint sum to compose activities at the same hierarchic level.

A HySChart can be seen as a tree with the primitive nodes as its leaves. The HySChart in Figure 3.17, for example, has node *Control* as its root and the nodes *inBend*, *inTol*, *up* and *down* as leaves. Starting from the tree's root, we derive the composed activity defined by the HySChart as follows (we write Act_N for the (primitive) activity of node N and $CAct_N$ for the composed activity of node N, here):

- if N is a primitive node, we define $CAct_N = da(Act_N)$, where $da(.)$ denotes the discontinuity adaption of activities (Section 3.2.3)
- if N has subnodes $M_1, \ldots, M_n$ which have the composed activities $CAct_{M_i} = +_{j=1}^{m_i} Act_{M_i,j}$, where each $Act_{M_i,j}$ stands for a sequential composition of adapted primitive activities, then we define:

$$CAct_N = +_{i=1}^{n} \left(+_{j=1}^{m_i} \left(da(Act_N) ;_+ Act_{M_i,j} \right) \right)$$

By convention we implicitly add activity $\frac{d}{dt} now = 1$ to the primitive activity Act_{root} of the root node. Provided *now* is initialized with zero, this ensures that it denotes the time since system start. The analog part is the composed activity of the root node of the HySChart, i.e. $Ana = CAct_{root}$. Figure 3.19 and the following example explain this definition.

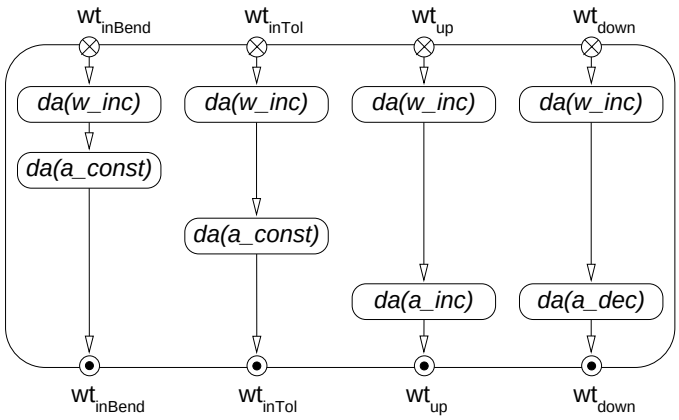


Figure 3.19: The *Control* component's analog part.

Example 3.8 (The analog part of *Control*.) The HySChart in Figure 3.17 has analog part:

$$\begin{aligned} Ana \;\equiv\; & (da(w\_inc) ;_+ da(a\_const)) + (da(w\_inc) ;_+ \mathsf{i} ;_+ da(a\_const)) + \\ & (da(w\_inc) ;_+ \mathsf{i} ;_+ \mathsf{i} ;_+ da(a\_inc)) + (da(w\_inc) ;_+ \mathsf{i} ;_+ \mathsf{i} ;_+ da(a\_dec)) \end{aligned}$$

where we applied associativity of $;_+$ and $+$, and used that the discontinuity adaptation of the identity again yields i. The activity names are used to refer to the semantics of each activity, here. Note that the expression is equivalent to $da((w\_inc ;_+ a\_const)) + da((w\_inc ;_+ a\_const)) + da((w\_inc ;_+ a\_inc)) + da((w\_inc ;_+ a\_dec))$ because the identity connector is the neutral element for sequential composition and $da(.)$ distributes over sequential composition. Figure 3.19 depicts the analog part as a graph. □

The entry and exit point symbols in Figure 3.19 highlight that the analog part has one path through the graph for every primitive node in the HySChart. When we construct the discrete part from the HySChart, we also get one wait entry and wait exit point at its highest level of hierarchy for each primitive node. This allows us to sequentially compose the discrete part with the analog part as in the semantics of a hybrid machine in Section 3.2. The distinct wait points allow both the discrete part and the analog part to know which node in the HySChart currently has control and to behave accordingly.

As we saw above the semantics definition of primitive activities ensures that the state-stream received and the produced output state-stream are equal. Regarding the definitions of sequential composition, disjoint sum of activities and discontinuity adaptation it is easy to see that this property is preserved by these operations. Thus, also for the analog part $((\iota, \sigma), \tau) \in Ana$ implies $\sigma = \tau$.

Resolvability. In Section 3.2 we demanded that the analog part is *resolvable*. This provides that it can react to any smooth external input and any start state received from the discrete part. As we have seen, the analog can be written as the disjoint sum

of sequentially composed, adapted activities. A sufficient condition for resolvability of the analog part is that all differential equations (and differential constraints) occurring in such sequentially composed activities are resolvable and the set of controlled variables occurring in each equation is disjoint from the controlled variables occurring in all other equations of the sequential composition. Resolvability of a differential equation (or constraint) means that the initial value problem the equation defines is solvable for any initial value and any smooth external input. Activities composed with disjoint sum may constrain the same variables, because only one of them determines the sum's behavior anytime. Resolvability may be lost if two sequentially composed solvable initial value problems have no solution valid for both.

3.5.3 Example: A Typical Hybrid Component

From the point of view of hybrid specification, the component *Filter* of the EHC system (Figure 3.2) is interesting. Its role is to eliminate noise in the input *sHeight* it receives from the environment. Furthermore, it allows the component *Control* to reset its internal state. The HySChart for the *Filter* component is depicted in Figure 3.20. Action name *set* stands for action $dReset? \wedge fHeight.val' = 0 \wedge fHeight.t' = now$ which resets $fHeight.val$ and updates the time stamp for the associated output channel *fHeight*. Activity name f_follow denotes the differential equation $\frac{d}{dt}fHeight = \frac{1}{T}(sHeight - fHeight)$, where T is the filter's time constant, i.e. a measure for its inertia. As invariant for state *Filter* we use the negation of the guard of *set*, i.e. $\neg(dReset?)$.

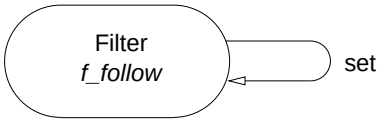


Figure 3.20: HySChart for the *Filter* component.

Both the discrete and the analog part of the filter are very simple. However, as there is a very close interaction of the discrete dynamics (the *set* action) and the continuous dynamics (the differential equation), it is hardly possible to decompose the filter into a purely discrete and a purely continuous component that cannot exhibit discontinuities. Therefore, the filter underlines the need for hybrid specification techniques.

3.6 Integration of Other Formalisms

The multiplicative interpretation of $\star$ not only allows to compose components specified under the additive interpretation (HySCharts), but it enables us to compose arbitrary components of type $\mathcal{I}^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_{p+}})$, where $\mathcal{I}$ and $\mathcal{O}$ is a set of input and output channels, respectively.

This means that components can be specified with any formalism which defines a component as a total, time guarded relation on piecewise smooth, piecewise Lipschitz continuous inputs and outputs. In particular this allows us to use description techniques from engineering disciplines, like e.g. block diagrams, which are widely used in control theory.

Example 3.9 (A delay element.) The delay element of the EHC transmits event *reset* with delay $\delta > 0$. By our conventions the event is encoded as a stream of time stamps, where the time stamp always gives the last point in time at which there was a discontinuity on the channel (Section 3.4). The output of the delay element has to satisfy the same convention since it also encodes events. Thus, if the input stream has a discontinuity as time u, its time stamp at that time also is u and the output of the delay element must have the same discontinuity at time $u+\delta$ with time stamp $u+\delta$. In other words, the time stamps in the (delayed) output stream must be incremented by δ to ensure that they correctly specify points of discontinuity. Further case distinctions are needed to treat initialization correctly. As a result, the delay element can directly be specified as a relation as follows:

$$
\begin{array}{lcl}
D & \in & {\mathbb{R}_+}^{\mathbb{R}_{p+}} \to \mathcal{P}({\mathbb{R}_+}^{\mathbb{R}_{p+}}) \\
D(\iota) & = & \{o \in {\mathbb{R}_+}^{\mathbb{R}_{p+}} \mid \forall u \in \mathbb{R}_+ . (u < \delta \Rightarrow o(u) = 0) \ \wedge \\
 & & \qquad (u \geq \delta \Rightarrow ((\iota(u-\delta) = 0 \ \wedge \ o(u) = \iota(u-\delta)) \ \vee \\
 & & \qquad\qquad (\iota(u-\delta) \neq 0 \ \wedge \ o(u) = \iota(u-\delta) + \delta)))\}
\end{array}
$$

In the general case of delay elements for arbitrary discrete or hybrid channels, the streams of time stamps are treated as above and the delayed stream of values τ is given by $\tau(u) = \sigma(u-\delta)$ for $u \geq \delta$ and stream of input values σ. In the case of delay elements for continuous channels we only need to consider the stream of values. □

3.7 Discussion and Further Work

3.7.1 Contribution

Based on a clear hybrid computation model, we were able to show that the ideas on interpretations of hierarchic graphs presented in [GSB98b] can smoothly be carried over to hybrid systems and yield modular, visual description techniques for such systems. Namely, the resulting techniques are HyACharts and HySCharts for the specification of hybrid system architecture and hybrid component behavior, respectively. A further motivation for using the extensive hierarchic graph framework as a semantic basis for HyCharts is the compositionality of these graphs w.r.t. refinement. Refining a node of a graph is guaranteed to result in a refinement of the whole graph. For HyACharts, this can be used directly. For HySCharts, the syntactic transformations which map them to hierarchic graphs must be observed. In more detail, this means that the refinement of actions, invariants and activities in a HySChart is guaranteed to result in

a refined HySChart, but some care is necessary to ensure that it defines a component which still is total in its input (see Chapter 4).

We demonstrated the use of HyCharts and their features with an example. Apart from many features which are common in Statecharts-like formalisms, HyCharts in particular offer the ability to compose HySCharts with components specified with other formalisms. In our opinion, such heterogeneous specifications are a key property for designing hybrid systems, as it allows the integration of description techniques from different engineering disciplines.

From a methodological point of view, we conceive a HySChart as an abstract and precisely defined mathematical model of a component in a hybrid system. Knowing the behavior of the analog part as specified by differential constraints allows us to develop more concrete models that can be implemented on discrete computers. This will be the topic of Chapter 5.

Although this thesis focuses on hybrid systems appearing in the context of disciplines such as electrical and mechanical engineering, we think that the continuous activities in HySCharts also make them well suited for specifying multi media systems, such as video-on-demand systems. Basically HyCharts seem to be appropriate for any mixed analog/digital system in which the use of continuous time is more natural than a discrete time model. In particular, this often applies to early development phases.

3.7.2 Related Work

The basic motivation for introducing these new description techniques were experiences obtained when modeling the EHC case study [SMF97] outlined above with hybrid automata [ACH+95]. A basic result of the case study was that the lack of modularity of hybrid automata complicates specification and analysis. As there is no concept of input and output variables in hybrid automata, the invariants of one automaton in a parallel composition may prevent transitions in parallel automata. Thus, the behavior of a hybrid automaton in a parallel composition cannot be studied independently of the others.[18] Furthermore, the lack of hierarchic states turned out to be inconvenient for specification. In this work, we developed a formal, modular description technique for hybrid systems that is associated with a visual formalism and incorporates advanced state machine features such as hierarchic states and preemption. In contrast to hybrid automata, HyCharts are modular and suitable for open systems.

The hybrid modules from Alur and Henzinger [AH97] are modular, but their utility suffers from the lack of a graphical representation and from the fact that it is not obvious how to model feedback loops. Parallel composition of modules is only possible if there are no cyclic dependencies between variables. At first sight this seems to

[18] For more details see [MS00].

prohibit feedback loops. However, loops are possible if the cyclic dependency is broken by a time step. For theoretical reasons, loops pose a problem in our approach too. We solve it by explicitly allowing feedback loops, as long as their well definedness is ensured. One way to do so is by including a delay in the feedback loop. By making this requirement explicit, users of our notation are guided to modeling loops correctly.

Another modular model, hybrid I/O automata, is presented in [LSVW96]. It is promising from the theoretical point of view, but it does not address some issues relevant for application in practice, such as graphical representation. In this respect, hybrid I/O automata, and also the hybrid modules mentioned above, are complementary to the work presented here. For HyCharts, aspects relating to the practical utility of the formalism are strongly emphasized.

A first approach towards a hybrid version of Statecharts can be found in [KP92]. The operational semantics given there, however, does not allow interlevel transitions and hierarchic specification of continuous activities. Therefore, this approach does not fully support hierarchy, unlike HyCharts, which permit both.

Except for HyCharts, all the models mentioned above are based on some kind of trace semantics in which continuous trajectories are pasted together at discrete time instances. At these instances, the preceding trajectory, the succeeding trajectory and possibly some intermediate discrete actions determine the values for the variables in the model. As the end point of the preceding trajectory, the values determined by intermediate discrete actions and the start point of the succeeding trajectory need not be equal, we find situations in which one variable is assigned a sequence of values at the same physical time instant. This means that such a trace is not isomorphic to a function of time. In our opinion this complicates combining the above models with models for continuous systems, which evolved in the engineering disciplines, because in such models the inputs and outputs of system components are usually functions of time. A decision must be made that determines which value of the variable is to be visible at a components interface at a physical time instant, i.e. we must find a mapping from the traces to functions. For hybrid automata, for instance, intermediate values in a sequence of discrete actions can cause further actions in parallel components. Thus, such a decision is hardly possible. For HyCharts, we use a simpler form of traces. Here, any variable in a component's interface has exactly one value at each time instant; the variable evaluation is a function of time. Note that the transition from traces to functions impedes the use of computational induction [Pnu94]. However, in Section 3.2.4 we saw how computational induction can be employed for components specified by HySCharts. Appendix A.1.2 formalizes this.

As already discussed in Chapter 2, commercial products for the design of embedded systems, like MATLAB/Simulink/StateFlow [TMI00], take a different approach to specifying hybrid systems. In this approach, the system needs to be partitioned into discrete-time or discrete-event state-transition-based components and components implementing control laws or continuous dynamics before specification can begin. While

this method is appropriate for the detailed design of many systems, we think it is inadequate for high-level design before particular control algorithms are fixed, and is highly inconvenient for specifying components that are hybrid themselves, like some environment models which contain, for example, phase transitions. A formal model that uses a specification approach similar to MATLAB/Simulink/StateFlow can be found in [EH96]. Interestingly there are some parallels between our hybrid machine model (Section 3.2) and the model presented there.

For logic and Petri net-based approaches to the formal specification of hybrid systems we refer the reader to [Lam93, CRH93, Wie96]. Apart from formal techniques, there is plenty of work on simulation packages for hybrid systems (see [Mos99] for an overview). Often these packages offer convenient graphical description techniques such as [WFSE96, Bro97a], but usually no formal semantics is defined for them. There also are simulation tools with a strong formal background [FvBR98, KFS95]. These, however, put less emphasis on visual specification. The ongoing work on the *Charon* system is an exception here, as it targets formal, visual specification as well as simulation [AGH+00, AGLS01]. The notation proposed there can be seen as a more pragmatic variant of HyCharts that builds on results obtained in the development of HyCharts. A further hybrid formalism which is related to Charon and was defined recently is *Masaccio* [Hen00]. It builds upon reactive modules and uses ideas for the formalization of control-flow which are similar to Charon and HyCharts.

To end this journey through the literature note that the graphical notation of HyCharts resembles the description techniques used in the software engineering method for (discrete) real-time object-oriented systems ROOM [SGW94]. In particular, the introduced HySCharts closely correspond to the hierarchic automata of ROOM, but extend them with continuous activities.

3.7.3 Further Work

Tool support is a major prerequisite for the application of HyCharts in practical systems development. [SPP01] formalizes the connection between the notations in the *MaSiEd* tool [AT98] to HyCharts. As a result, *MaSiEd* can be regarded as a simulation and modeling tool for a dialect of HyCharts. Modeling support and simulation, however, could still be improved. In particular, in the tool a syntax for action guards and bodies with a leaner, more precise semantics is desirable. Currently actions are defined with C++ code in *MaSiEd.* Furthermore, automatic verification of HyCharts is desireable. We believe that the techniques known for linear hybrid automata [ACH+95] can easily be adapted. Although one must be aware of the severe efficiency limits of these algorithms, they can nevertheless be used beneficially in specific circumstances, where the continuous dynamics is easy and the overall system is not very complex [MS00].

Chapter 4

Notes on Refinement in Continuous-Time

Besides refinement affecting the time model which is discussed in Chapter 5, refinement rules for HyCharts which do not affect the time model are also an important element of the infrastructure needed for the integrated development process of Chapter 2. Such rules are applied in the early development phases, before the system is partitioned into discrete-time and continuous-time subsystems. There, they help to formally establish the traceability of requirements from first abstract models to more concrete ones, if they are used for model transformations [Bro97c]. As this class of refinement rules is not substantially different from stepwise refinement rules for discrete systems, we do not put great emphasis on them in this thesis. Instead the thesis focuses on refinement affecting the time model (Chapter 5). Thus, this chapter is mainly intended to demonstrate the analogies to refinement rules for discrete systems, without going into formal detail, and to make the reader more familiar with refinement in the context of HyCharts. It may be skipped by readers not interested in refinement of HyCharts in a continuous time model.

4.1 Preliminaries

Refinement. In Section 3.3.5 refinement of relations was introduced on basis of set inclusion. For two relations A and B, A is a refinement of B, written as $A \preceq B$, was defined as meaning that $A \subseteq B$ holds. If A and B are components as defined by HyCharts, i.e. if they are input/output relations over dense streams, $A, B \subseteq \mathcal{I}^{\mathbb{R}_{p+}} \rightarrow \mathcal{P}(\mathcal{O}^{\mathbb{R}_{p+}})$, refinement of B by A means that the behavior of A is contained in that of B. Thus, A may only be more precise than B. Every behavior (ι, o) of A, $o \in A(\iota)$, also is a possible behavior of B.

Properties which constrain all possible behaviors of a system clearly are maintained by refinement. Broy therefore also calls this refinement notion *property refinement* [Bro97b, Bro01]. Note that a detailed study of properties of hybrid systems is given in Chapter 6 where the effect of refinement on these properties is also discussed. In the development of systems not only the behavior, but also the interface of a system is sometimes modified. [Bro97b] therefore introduces the more general notion of *interaction refinement*. We will only consider two cases of interaction refinement, called *U-simulation* and *U^{-1}-simulation*, in this thesis. In these special cases an input/output relation $A \subseteq \mathcal{I}_a \to \mathcal{P}(\mathcal{O}_a)$ is an interaction refinement of $B \subseteq \mathcal{I}_b \to \mathcal{P}(\mathcal{O}_b)$, if A adapted to the type of B is a property refinement of B (U-simulation), or—the other way round—if A is a property refinement of B adapted to the type of A (U^{-1}-simulation). Formally, two pairs of a concretization relations and abstraction relations are needed here:

$$\begin{array}{lcllcl} c_1 & \in & \mathcal{I}_b \to \mathcal{P}(\mathcal{I}_a) & a_1 & \in & \mathcal{I}_a \to \mathcal{P}(\mathcal{I}_b) \\ c_2 & \in & \mathcal{O}_b \to \mathcal{P}(\mathcal{O}_a) & a_2 & \in & \mathcal{O}_a \to \mathcal{P}(\mathcal{O}_b) \end{array}$$

where the sequential composition of concretization and abstraction relation is the identity relation, i.e. $c_1 ;_\times a_1 = \mathsf{I}_{\mathcal{I}_b}$ and $c_2 ;_\times a_2 = \mathsf{I}_{\mathcal{O}_b}$ ($;_\times$ and I are defined in Section 3.3.2). Then, A is an interaction refinement of B, if one of the following holds:

- $c_1 ;_\times A ;_\times a_2 \preceq B$ *(U-simulation)*
- $A \preceq a_1 ;_\times B ;_\times c_2$ *(U^{-1}-simulation)*

Note that these are only two of four variants of interaction refinement as defined in [Bro97b]. However, we will not use the others in this thesis. We need interaction refinement in the remainder of this chapter when relations with different types are compared.

Graph algebra. The following section discusses if and how the refinement rules presented in [PR97, PR99] can be carried over to HyCharts. As it will be outlined, many of these general rules directly follow from axioms which hold for the interpretations of the graph algebraic framework that serves as foundation for HyCharts (see Section 3.3). For an overview of these axioms the reader is referred to [Şte94] and in particular to [GBSS98], where the axioms are explained in detail and depicted graphically. The axioms include associativity of sequential composition, commutativity and associativity of visual attachment and further axioms which allow us to shift nodes on feedback and which relate feedback with sequential composition and visual attachment. As described in [GBSS98] the axioms can be used for design transformations, like optimization.

4.2 Architecture Refinement

Philipps et al. introduce various rules for the refinement of the architecture of discrete systems in [PR99] and [PR97]. In detail, the authors provide two central rules for

the isolated refinement of components of a system architecture. Furthermore, a set of auxiliary rules is introduced which allow the designer to add and remove channels and components, and to expand and fold definitions of hierarchic components, which themselves are specified with architecture descriptions. In this section we discuss these rules in the context of HyACharts.

Refinement of components. Let $G[C]$ be the multiplicative hierarchic graph for a HyAChart which contains a (primitive or hierarchic) component C. Then, compositionality of (multiplicative) hierarchic graphs w.r.t. refinement, ensures that $G[C']$ is a refinement of $G[C]$ if $C' \preceq C$. In [PR99], a similar compositionality property is formulated as an inference rule. Additionally, the paper defines a further rule for component refinement in presence of a global condition. Basically this rule allows the designer to infer that $G[C']$ is a refinement of $G[C]$, if C' is a refinement of C under condition p and p holds for $G[C]$. As we do not want to formalize global conditions here, we do not formally transfer this to our context.

Auxiliary rules. Due to the different encoding of system architecture used in [PR99] there is no direct correspondence of the rules given there to possible manipulations of HyACharts. We therefore present the underlying ideas of the auxiliary rules and discuss similar manipulations of HyACharts. The rules provided in [PR99] allow the designer to add output channels to components such that the output on the channel is completely chaotic and they allow to delete output channels that are not used. The rules for the input channels allow the designer to add input channels to a component such that the input is ignored and they allow removing input channels if the input is not used. The rules for components provide that components which are not connected to the other components of the system architecture and to the environment may be added or removed. For HyACharts, similar rules can be defined by using the neutral connectors $\curlyvee^0$ and $\curlywedge_0$ of multiplicative hierarchic graphs (see Section 3.3.2).

As an example, we regard the addition of an input channel to a component C such that the input channel is ignored. On the graph level, the fact that the input is ignored in the new component corresponds to the behavior of the connector $\curlywedge_0$, which is related to projection (Section 3.3.2). Thus, the new component can be defined by $C' \equiv \curlywedge_0 \times C$. We now prove that C' is an interaction refinement of C. As concretization relation from the input space of C to the input space of C' we use $\curlyvee^0 \times \mathsf{I}$. Its corresponding abstraction is $\curlywedge_0 \times \mathsf{I}$. The abstraction relation from the output space of C' to that of C is the identity. The corresponding concretization also is the identity. It then remains that show that $(\curlyvee^0 \times \mathsf{I});_{\times} (\curlywedge_0 \times C) \preceq C$ holds. This is an immediate consequence of the axioms for hierarchic graphs given in [GBSS98] which even provide equality. This result is depicted graphically in Figure 4.1, where $\vdash$ denotes $\curlyvee^0$ and $\dashv$ denotes $\curlywedge_0$.

Two further auxiliary rules in [PR99] allow expanding or folding the definitions of hierarchic components. They strongly depend on the formalization of system architecture used there. In contrast to [PR99], the semantics of hierarchic graphs does not refer to component and channel names, but rather to the relativ position of channels

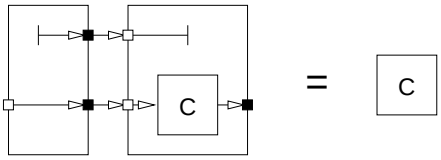


Figure 4.1: Adding of (ignored) input channels.

(arcs) and components (nodes) and to the way they are connected. Names only are "decoration" in HyACharts. In HyACharts expanding and folding therefore merely corresponds to applying the associativity of the graph operators.

Note that the utility of the auxiliary rules for practical examples, like the one given in [PR99], stems from their interaction with the rules for the refinement of components.

4.3 Behavior Refinement

In this section we first discuss the refinement of actions, invariants and activities in HySCharts. Then, the rules for adding and removing transitions given in [Sch01] are studied in the context of HyCharts. Adding of states and the hierarchic refinement of states, which is also defined in [Sch01], is not considered in direct analogy to that work. Instead, we outline how, and under which conditions, the extension of a HySChart by further control states, variables, input and output channels can be regarded as (interaction) refinement. This is also contrasted to the hierarchic refinement technique presented in [AGLS01] for the related formalism *Charon* [AGH+00].

Refinement of actions, invariants and activities. As starting point we consider a HySChart with fixed actions, invariants and activities given. Let $a, a' \subseteq (\mathcal{I} \times \mathcal{S}) \times \mathcal{S}$ be two actions or invariants such that a' results from strengthening a, i.e. $a' \subseteq a$ holds. The graph transformations which lead from the graph for a HySChart to the additive hierarchic graph which defines the HySChart's semantics does not negate or otherwise modify the actions and invariants (Section 3.5.1). Thus, the discrete part of a HySChart which contains action or invariant a is a additive hierarchic graph containing relation a as a primitive node. Due to monotonicity of additive hierarchic graphs w.r.t. set inclusion (Section 3.3.5), we are guaranteed that replacing a by a' results in a discrete part which is a subset of the original one. If the new discrete part is total, the resulting HySChart is a refinement of the original one, because the machine model of HySCharts is defined by a multiplicative hierarchic graph (see Figure 3.4, middle) and therefore compositional w.r.t. refinement. If it is not total, the subset relation still holds, but the resulting HySChart is not a sound input/output relation, since it is not total. Conditions on actions and invariants which ensure totality of the discrete part were already discussed in detail in Section 3.5.1. A sufficient condition e.g. is that for every primitive and hierachic node in the chart and for any input and

latched data state, at least one action guard of an outgoing transition or the node's invariant is true. Furthermore, all actions in a sequential composition of actions must be enabled if the first one is.

We stress that for this refinement we consider actions and invariants in a HySChart to be fixed. This means that after a refinement step in which actions are modified we do not derive new invariants from the actions, but continue to use the old ones. This is important, because invariants are often derived from action guards by negating them (cf. Section 3.5.1.5). Informally, the outlined kind of refinement reduces the nondeterminsm w.r.t. when and under which conditions a state may be left.

For the refinement of activities, the situation is similar. The transformation of the graph for a HySChart to the time-extended hierarchic graph for the analog part also does not negate or otherwise modify the activities. Due to monotonicity of time extended additive hierarchic graphs w.r.t. set inclusion (Section 3.3.5), refining an activity in a HySCharts results in the refinement of its analog part. The new HySChart is a legal component and a refinement of the old one, if the refined analog part still is resolvable (Section 3.5.2). For instance, in the HySChart for the EHC (Figure 3.17) this immediatly allows us to replace activity $a\_inc \equiv \frac{d}{dt} aHeight \in [cp_-, cp_+]$ by $a\_inc' \equiv \frac{d}{dt} aHeight = cp_-$ in order to obtain a refinement of the original HySChart.

Adding and removing transitions. In [Sch01, Sch98] a number a refinement rules are defined which allow the designer to refine component specifications which are given in a Statechart dialect called μ-Charts [PS97]. Among others, these rules define conditions under which transitions in the diagram may be added and removed.

According to [Sch01] an outgoing transition of a node in a μ-Chart may be removed in a refinement step, if the action guard is already subsumed in the action guards of the other transitions emerging from the same node. For HySCharts, this is also valid and can be reduced to the refinement of actions as desribed above.

Furthermore, in the μ-Chart formalism a transition may be added in a refinement step under certain conditions. This can be explained as follows. In contrast to HySCharts, μ-Charts use a *chaos-semantics*. This means that if at a time instant no transition is enabled (μ-Charts do not contain invariants), the behavior of the component from then on is completely chaotic, i.e. its possible output from then on is the whole output space. In the sense of HySCharts, the condition for adding transitions in μ-Charts means that the action guard of the added transition must be disjoint from the action guards and the invariant of the node to which it is added. For μ-Charts, this is a valid refinement, because one case in which chaotic behavior may occur in the original chart is eliminated this way by specifying more detailed behavior. In the context of HySCharts this would not be a valid refinement. As HySCharts do not use a chaos semantics, the behavior of the original chart would be empty in cases where the action guard of the transition to be added is true. However, after the addition it would be nonempty. Choosing the sharper semantics for HySCharts was motivated by the observation that it corresponds closer to the semantics which is implemented in

design tools for state transition diagrams, like e.g. in the AutoFocus tool [BHS99]. Thus, the price for this closeness is less flexible refinement.

Hierarchic refinement. In this paragraph we examine under which conditions an extension of a HySChart's input space, output space and state space results in (interaction) refinement. In particular, this includes the case where a new level of hierarchy in introduced by splitting a node in the chart into further subnodes. Let h_1 and h_2 be two HySCharts with input spaces $\mathcal{I}_i$, output spaces $\mathcal{O}_i$ and state spaces $n_i \cdot \mathcal{S}_i$, $i \in \{1, 2\}$. We require that $\mathcal{I}_1$ is a factor space of $\mathcal{I}_2$, i.e. $\mathcal{I}_1 = \pi_{\mathcal{I}_1}(\mathcal{I}_2)$ must hold, where $\pi_{\mathcal{I}_1}$ denotes Cartesian projection on space $\mathcal{I}_1$. In other words, $\mathcal{I}_2 = \mathcal{I}_1 \times \mathcal{I}'$ holds (up to isomorphism) for some $\mathcal{I}'$. Note that this means that h_2 has more input variables than h_1, because the spaces are defined as the product of the variable domains (Section 3.2.1). Furthermore, $\mathcal{O}_1$ must be a factor space of $\mathcal{O}_2$, $\mathcal{S}_1$ must be a factor space of $\mathcal{S}_2$, and $n_1 \leq n_2$ must hold. This last condition expresses that the control-state space of h_2 is larger than that of h_1.

For interaction refinement we first need some concretization and abstraction relations. We define two generic relations as follows. For factor space Y of space X, we define $a_{X,Y}(x) = \{\pi_Y(x)\}$, where π_Y again denotes Cartesian projection on Y.[1] Furthermore, the concretization is defined by $c_{Y,X}(y) = \{x \mid y \in a_{X,Y}(x)\}$. The abstraction from $n_2 \cdot S_2$ to $n_1 \cdot S_1$ is defined by $a_{21}((k, s)) = \{(ren(k), \pi_{S_1}(s))\}$, where ren is a function from $\{1, \ldots, n_2\}$ to $\{1, \ldots, n_1\}$ which is onto. Function ren corresponds to the renaming and identification of the control states of h_2. It reflects that control states in HySCharts are identified by a number, not by a name. The concretization relation corresponding to a_{21} is defined by $c_{12}((k, s)) = \{(k', s') \mid (k', s') \in a_{21}((k, s))\}$. As usual, the extension of these relation to dense streams is denoted by marking them with †. It is easy to see that for all of these relations, concretization sequentially composed with abstraction is the identity relation.

As a consequence of this property, a sufficient condition for the interaction refinement of h_1 by h_2 is that the discrete part, the analog part and the output projection of h_2, denoted by Com_2, Ana_2 and Out_2, are interaction refinements (actually U^{-1}-simulations) of those of h_1, denoted by Com_1, Ana_1 and Out_1. Formally this condition is expressed as follows:

$$\begin{array}{lcl} Com_2 & \preceq & (a_{\mathcal{I}_2,\mathcal{I}_1} \times a_{21}) \,;_\times Com_1 \,;_\times c_{12} \\ Ana_2 & \preceq & ({a_{\mathcal{I}_2,\mathcal{I}_1}}^\dagger \times {a_{21}}^\dagger) \,;_\times Ana_1 \,;_\times {c_{12}}^\dagger \\ Out_2 & \preceq & a_{21} \,;_\times Out_1 \,;_\times c_{\mathcal{O}_1,\mathcal{O}_2} \end{array}$$

Basically this results from introducing the identity relation ${c_{12}}^\dagger \,;_\times {a_{21}}^\dagger$ at various position in the graph of the hybrid machine model (Figure 3.4, middle) and cutting the graph into appropriate pieces.

[1] Projection is overloaded to arbitrary spaces of which Y is a factor space. Therefore, X does not occur as index in π_Y.

A sufficient condition to ensure that these refinement relations are valid is that h_2 results from hierarchically refining a node in h_1 by a graph which only modifies those variables in the state space of h_2 which are unknown in h_1. Because of U^{-1}-simulation as above, the behavior of h_1 adapted to the interface of h_2 is completely chaotic on the new parts of the state space. Thus, h_2 may specify arbitrary behavior for the new part, as long as the behavior w.r.t. the old part of the state space coincides with that of h_1.

As an example, let us assume that HySChart h_1 is given by Figure 3.17, left, without its substates and without variable w and activity *w_inc*. The entire HySChart of Figure 3.17 with all its hierarchic levels, but with activity *w_inc* as an activity of the nodes *inTol* and *outTol*, is regarded as h_2. Under these conditions h_2 is an interface refinement of h_1. This holds, because h_2 stems from extending h_1 by channels/variables *fHeight*, w, *reset* and *aHeight*. The behavior w.r.t. inputs on channel *bend* does not differ between h_2 and h_1. Activity *w_inc* had to be removed from Figure 3.17, left, for the refinement, because otherwise w would already be a variable of h_1 and its evolution therefore would already be constrained.

As we have seen, a consequence of the above general rule is that a node may be refined to a hierarchic node, if the added subgraph only modifies variables which are not in the state space of the original HySChart. [Sch01] defines a related rule for μ-Charts which allows the designer to add hierarchy, if a similar condition holds and if the triggering of hierarchic transitions corresponds to weak preemption, which is the variant of preemption used in HySCharts (see Section 3.5.1.4).

As far as extension of the state space is concerned the above general rule corresponds to a similar rule defined for *Charon* in [AGLS01]. *Charon* supports this kind of refinement in an easier manner, because there the program-state state in not a flat disjoint sum as for HySCharts, but a stack of disjoint sums of local data-state spaces. The stack directly corresponds to the hierarchy in the visual representation and the *push* and *pop* operations on the stack correspond to abstraction and concretization, respectively.

4.4 Discussion and Further Work

We have discussed which refinement techniques for system architecture as introduced in [PR99] can be carried over to HyACharts. Furthermore, we outlined possible refinement rules for state machines and related them to the rules given in [Sch01]. In particular for the system architecture, it turned out that the axioms described in [GBSS98] can directly be employed to provide the semantic foundation for standard transformations of system architecture as defined in [PR99]. This provides further evidence for the utility of using the algebraic framework as semantic foundation of HyCharts.

Related work. The immediately relevant related work concerning the refinement of system architecture [PR99] and component behavior [Sch01, AGLS01] has already been extensively discussed in this chapter. Stepwise refinement of state machines is also studied in [Rum96] in a way similar to [Sch01]. Further pointers to the literature on refinement in particular in the context of hybrid systems are given in Section 5.7.2.

Further work. For future work, a stricter formalization of the outlined results on the hierarchic refinement of HySCharts is desirable.

Chapter 5

Refinement and Time: From Continuous-Time to Discrete-Time

Moving from an abstract model based on a continuous time scale to implementation amounts to changing to a discrete-time execution scheme for those components in the model which are implemented on (or in) digital hardware. Due to the cost structure and flexibility of software-based solutions, such discrete-time execution usually is desired for large parts of a hybrid system. In order to ensure that vital properties of the abstract model are satisfied in the implementation oriented discrete-time model, the change of the execution scheme must be performed in a controlled way. This is an essential prerequisite for the utility of an integrated development process as described in Chapter 2. Therefore, this chapter identifies conditions under which the transition from continuous-time to discrete-time is a formal refinement in the sense of Section 3.3.5. Chapter 5 will show that the employed notion of refinement indeed maintains vital properties of hybrid systems.

Two basic elements are necessary for the transition from HyACharts with components operating in continuous-time to HyACharts containing subcomponents that operate in discrete-time. First, we have to clarify how primitive components specified with HySCharts can be refined to a discrete-time execution. Second, we have to define how the obtained discrete-time components are integrated into architecture diagrams operating in continuous-time, and give further techniques which allow us to directly combine discrete-time components, without transferring their output to continuous-time and discretizing is again.

Overview. This chapter is organized as follows. First, we explain effects of an execution based on sampling and their implications on HySCharts in Section 5.1. This motivates the introduction of a more flexible version of the machine model from Section 3.2 and a strategy for the systematic usage of state invariants in HySCharts (Section 5.2). As a notation for the discrete-time version of HySCharts we introduce *DiSCharts* in Section 5.3. They are very similar to HySCharts, but differ from them in

their underlying discrete-time machine model. With this basis Section 5.4 presents a refinement technique for the transition from (continuous-time) HySCharts to (discrete-time) DiSCharts and demonstrates it with an example. Section 5.5 introduces further techniques to couple discrete-time components within a HyAChart. As a side remark Section 5.6 explains how the discrete part of a HySChart can be implemented in discrete-time while the time model for the analog part remains unchanged. Finally, we discuss the results of this chapter and give a summary of related work in Section 5.7.

5.1 Methodological Aspects

Implementation effects on state transitions. In our view hybrid automata-like notations, such as HySCharts, can beneficially be used for requirements capture and the early design steps of hybrid embedded systems. In these phases designers usually want to express that some actions are taken when certain conditions are satisfied. They are not so much interested in the detailed timing, i.e. in possible small delays between the (first) satisfaction of conditions and the execution of the corresponding actions. They are, however, aware that such delays exists when the model is implemented on (or in) digital hardware. One primary reason for such delays is that the digital components can only sense the status of their environment within the discrete time grid given by their clock. While we think it is not adequate to already consider sampling rates and timing uncertainties in detail in early design phases, models must nevertheless be designed in a way that tolerates small delays without violating vital system properties. Otherwise, if the model's correctness relies on the absence of any delays, it cannot be implemented later on.

Besides timing uncertainties, there is a further kind of deviations from the ideal model. When a transition which depends on the detection of a boundary crossing of a continuous variable is taken by a digital component, the boundary will usually already be exceeded. In other words, in case of transitions which depend on the value of variables that evolve continuously, the timing uncertainty corresponds to an uncertainty in the actual value of the variable for which the transition is taken. For given analog dynamics, one kind of uncertainty can be derived from the other.

Figure 5.1 visualizes the effect of sampling on the detection of a boundary crossing for boundary value c. For trajectory x and sampling period T, the boundary crossing of x is detected with delay. The crossing of trajectory y is not detected with sampling period T, because it happens between two consecutive sampling instants. Trajectory z only hits the boundary at a single point in time. This also is not detected with the used sampling period. Trajectories y and z thus indicate that the boundary crossing cannot be detected with arbitrary precision with sampling. For HySCharts employing such boundary crossing events as action guards, this means that transitions will usually not be taken immediately when enabled, but with some delay or, under

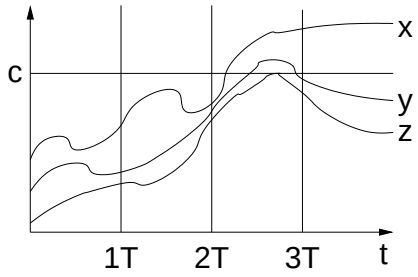


Figure 5.1: Delayed or failing detection of boundary crossing events caused by sampling.

awkward circumstances, not at all.[1] Considering the value of the trajectory when such a transition is taken, we see that in the depicted example x is already greater than c at $3T$. In the ideal case, the transition would be taken for $x(t) = c$. Obviously, the delayed execution of transitions may violate invariants which are required to hold for the system or may lead to instability, because of too late reaction.

Implementation effects on analog dynamics. Let us now regard the analog dynamics of a hybrid component. In terms of HySCharts, this corresponds to regarding the output of the analog part of the machine model (cf. Figure 3.4, middle). This output on the one hand affects the discrete part and on the other hand it can affect external components. We assume the dynamics are given by some differential equation $\dot{x} = f(x, i)$, possibly depending on external input i. In this case we distinguish two kinds of effects resulting from an implementation of the analog dynamics in discrete time. At sampling instants the discretized analog dynamics in general only yields an approximation of the exact dynamics. As an analogy, think of a numerical algorithm (the discretized analog dynamics) that computes the solution to $\dot{x} = f(x, i)$ for given input i and initial value x_0 (the exact dynamics). At each interpolation point kT of the algorithm, it will provide an approximation of the exact value of the solution $x(kT)$. Like in numerics, we call the deviation between the exact value and the approximating value at sampling instants the *discretization error* [Sch88]. It affects the discrete part as well as external components. A further error results between sampling instants, where the discretized analog dynamics does not produce new output. In this thesis, we choose to extend the (discretized) analog dynamics' output at a sampling point over the interval until the next sampling instant by holding it constant. In control systems terminology, this correponds to a so-called *zero-order hold*. The effect of this strategy is that there is a further deviation of the output w.r.t. the exact dynamics between sampling instants. At the sampling points this deviation however vanishes. We therefore call it the *intersample error*. Both kinds or errors are visualized in Figure 5.2. We define the intersample error as the maximum deviation between the

[1] In fact it is a major task in controller design to ensure that any important event, such as a boundary crossing, is detected by the controller.

exact value and the time-extended output of the discetized analog dynamics during a sampling period. In HySCharts, this error affects the discrete part, because between sampling points the exact dynamics could enforce a transition, while no transition may be necessary for the discretized dynamics which simply remains constant during a sampling interval. Furthermore, the environment of a component is affected by this error.

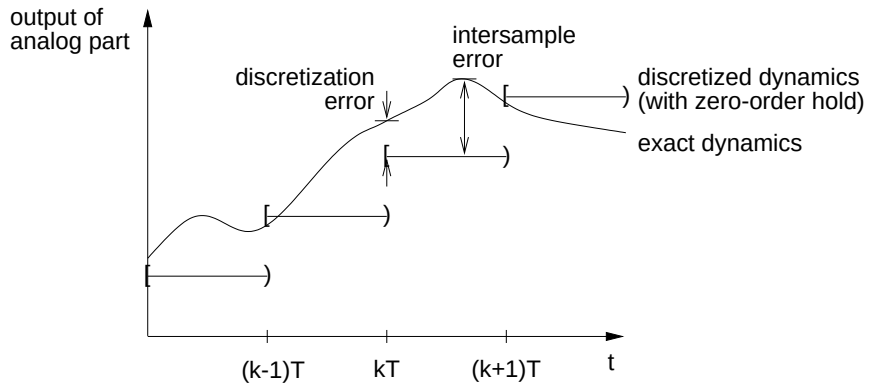


Figure 5.2: Discretization error and intersample error.

Other extension strategies besides the zero-order hold would be feasible as well, but, as Ogata mentions, more complicated construction schemes for continuous-time signals from discrete-time signals are usually not used in control applications [Oga87]. The reason is that more complicated schemes usually involve a delay in construction of the signal. Such delays can result in inappropriately late reactions and can lead to instability of the system (Section 6.3.3 introduces stability in detail). In the context of digital signal processing, more complicated so-called *reconstruction filters* are used to obtain a smooth, analog output signal from a digital input [Smi97]. In any case, considering the intersample error separately in principle allows us to also consider such reconstruction schemes.

Impact on verification. The effect of discrete time implementation also has to be considered when planning verification steps for the system. Namely, there is not much sense in formal, exhaustive verification of abstract models with tools like model checkers, if the properties established for the abstract model can not be transfered to the implementation, because the abstract model is unrealistically precise. In contrast, verification is helpful if the abstract model is liberal enough to comprise effects that result from its implementation, like time delays and deviations from the exact continuous dynamics. Model checking techniques which weaken the exactness with which analysis of hybrid models is performed have been proposed in [GHJ97, HR00] and [Frä99] (see Section 5.7.2).

Methodology. As a result, the models of a system component specified in the

requirements capture phase and the early design phase should allow for timing uncertainties and uncertainties in the analog dynamics, i.e. uncertainties in the exact value of a component's real-valued variables at a given point in time. Similarly, uncertainties in the exact value of input signals should also be allowed for, since the input may come from external components which are also subject to errors. From a methodological point of view we propose to first specify system components with eager transitions[2] and exact analog dynamics, keeping in mind that delays exist in practice, and that values cannot be detected and computed with arbitrary precision. Then, with some additional input from the designer, a modeling tool can automatically create a *relaxed* model which allows that transitions are taken with some delay and that there are deviations from the exact analog dynamics. This relaxed model requires a notation that incorporates delayable transitions and therefore motivates invariants in our HySCharts. For the specification of analog dynamics with permitted discretization and intersample errors we will explain below, how they can be integrated into HySCharts.

In further design steps the timing uncertainty and the permitted deviations from the analog dynamics can be used by the designers to select a sampling rate for the embedded hard- and software that guarantees (1) that transitions are taken within the delay permitted by the relaxed model and (2) that the discretized analog dynamics is sufficiently precise. The selection process can be guided by the modeling tool and may involve standard techniques from control theory, like Shannon's theorem (Section 2.1). Notations with delayable transitions are useful in this context because they allow us to regard the step to a model with a fixed sampling rate as a refinement step. Note that sampling rate selection is constrained by the dynamics of the underlying physical environment and the functionality required from the overall system. It is not clear from the beginning, except if the dynamics is already well known from legacy systems, whose only part to be updated is the software.

Alternatives. An alternative to this methodology would be to establish ideal properties for an ideal model first and then move to a relaxed model comprising implementation effects such that a relaxed version of the ideal properties is guaranteed to hold for the relaxed model. This is promising in principle, because working with ideal models is often easier than working with relaxed models. However, in general this is not possible. As an example, let us assume an (ideal) HySCharts whose transitions are taken as soon as they are enabled. If we relax the model and permit that transitions are taken with a small delay, the resulting output traces possibly not only deviate a little from the HySChart's ideal output traces. Qualitatively different behavior is also possible, because there may be control states which are unreachable in the exact model due to an infinitely fast response or due to arbitrary precision in this model. In other words, the problem is that arbitrarily small, non-zero deviations from the

[2]Remember that transitions are called *eager* if they are taken as soon as their guard is true (Section 3.5.1.5).

exact values of variables as well as arbitrarily small delays in the taking of transitions can trigger (further) transitions which can never be taken in the exact model. For instance, in Figure 5.1 for trajectory x a further transition could be enabled at time $3T$ due to the value of x being so much above c.

The situation would be the same, if we allowed that enabled transitions are taken immediately in the ideal as well as in the relaxed model. In this case, errors in the analog dynamics could still trigger transitions which would never be enabled in the ideal model.

5.2 Relaxed HySCharts

As indicated in Section 5.1, we cannot hope to detect the exact moment in time when an action guard in a HySChart becomes true, if we only evaluate it in a discrete-time manner. Similarly, a discrete-time version of the analog part defined in a HySChart does not result in the same behavior as the original analog part, but only in an approximation of it (see Figure 5.2). In order to carry over properties from one system to another our refinement notion requires that the set of possible behaviors of the refined system be a subset of those of the original system. When using a very precise semantics for action guards and activities in the specification of the original system there is not much hope in finding useful refined systems. We therefore propose a way to specify relaxed models with HySCharts which do allow flexible refinement.

The relaxation we propose is motivated by two observations, as indicated in Section 5.1. First, for the discrete part sampling typically results in transitions being taken some time after they have become enabled. Second, for the analog part a discretization error and an intersample error arises from sampling. We define the HySChart relaxation such that it comprises bounds for these kinds of errors. A first approach to do so is to include the permitted discretization error and the permitted intersample error in a relaxed analog part such that the discrete part also is directly affected by them. However, we prefer a less liberal relaxation which is closer to the HySChart with the exact semantics, because it generates less relaxed behavior. When a sampling implementation for a HySChart is constructed, this can be done by choosing the same sampling instants for the discrete part as well as for the analog part. In this case, the discrete part is directly affected only by the discretization error. The intersample error, however, can only cause that the discrete part fails to take a transition on time. This effect can already be considered together with the delayed taking of transitions. It cannot cause that the discrete part takes a (wrong) transition due to the intersample error in some variable's value, because the discrete part is only active at sampling instants where the intersample error by definition vanishes. Nevertheless, the intersample error is visible at the output interface of a relaxed HySChart. Therefore, we only include the permitted discretization error in the analog part and add the permitted intersample error at the output interface of a relaxed HySChart.

The corresponding machine model is introduced in Section 5.2.3. Using this tighter relaxation is motivated by the hope that proving properties for the tight relaxation is easier than for the coarser one, because we have less nondeterministic behavior. On the other, hand finding implementations for the tighter relaxation is not expected to be more difficult than for the more liberal one, since in both cases the same error bounds can be used to find a discretization of a given analog part.

5.2.1 Constructing Relaxed Invariants

Choosing the invariant of a state in a HySChart as the negation of the disjunction of the actions guards of the transitions directly emerging from a node (and from none of its subnodes), as proposed in Section 3.5.1, results in eager transitions. We call such an invariant an *exact invariant*. In presence of exact invariants transitions must be taken as soon as their guard is true. As explained, this is too restrictive for refinement and not realistic. Starting from such an exact invariant, the method described in the following therefore relaxes this invariant by allowing that the relaxed invariant remains true for small deviations of the variables' values from those allowed in the exact invariant and by allowing that is also remains true for some time after discrete changes in the variables. Otherwise such discrete changes could immediately violate the exact invariant and enforce a transition.[3] If the designer specifies the permitted deviations according to his or her understanding of the problem, the relaxed invariant can automatically be constructed from the exact invariant by a CASE tool.

Preliminaries: Action guards. Before we define relaxed invariants some details of the syntax of action guards have to be defined. The semantics of actions is discussed in Section 3.5.1.3. Syntactically we define an action guard as a conjunction of atoms of the form $p(d.\vec{val}', \vec{v})$, $a(h.\vec{val}', c.\vec{val}', \vec{v}) \natural\ 0$ and $ts' \neq ts$, where $d.\vec{val}'/h.\vec{val}'/c.\vec{val}'$ are tuples of variables denoting the value of current input on discrete/hybrid/continuous input channels, $\vec{v}$ is a tuple of controlled variables excluding latched time stamps, $\natural \in \{>, \geq, \neq\}$, and ts' is the current time stamp of an input channel and ts the corresponding latched time stamp. Symbol p stands for propositions over variables with domains different from the real numbers and a refers to arithmetic expressions. Such propositions and operators for building arithmetic expressions are assumed given. Expressions of the form $x\#c$, where $\# \in \{\geq, \leq, =, >, <, \neq\}$, for a real-valued variable x and constant c, which often occur in practice, can be reduced to the above normal form. Atoms of the form $ts' \neq ts$ are needed to detect discrete jumps (or events) on input channels. Note that time stamps and other real valued variables must not be mixed in action guards. Formally, guard syntax is defined by:

[3] The dual approach to relaxing the exact invariant would be to relax the exact action guard. However, this would be greater technical effort for HySChart, because relaxing a guard which depends on the presence of an event cannot simply be defined by allowing the guard to become true *before* the event. In contrast, allowing the guard and the invariant to remain true after the event, which is done here, is sound and straightforward.

$$g ::= p(d.\vec{val}^{\backprime}, \vec{v}) \mid a(h.\vec{val}^{\backprime}, c.\vec{val}^{\backprime}, \vec{v}) \not\downarrow 0 \mid ts^{\backprime} \neq ts \mid g \wedge g$$

To allow derivation of the sampling rate in Section 5.4.2 we furthermore require that the evolution of arithmetic expressions be Lipschitz continuous provided the evolution of the variables occurring in them is. For simple arithmetic expressions, like those merely involving sums and products, this holds.

Relaxed invariants. As indicated in Section 3.5.1, for a state in a HySChart, its exact invariant results from the negation of the disjunction of all the guards of transitions emerging directly from the node (and from none of its subnodes). This is equivalent to the conjunction of the negated guards. We therefore assume the exact invariant to be given as a conjunction of negated guards. Furthermore, we only allow non-strict inequalities as comparison operators between arithmetic expressions which do not contain time stamps. Together with using the discrete topology for time stamps and for variables with domains different from $\mathbb{R}$ (cf. Section 3.5.1.5), this ensures that the guards are topologically closed sets which in turn guarantees that the exact invariants and the resulting relaxed invariants are open. This is needed for well definedness of the hybrid computation model (Section 3.2.2). We do not define the formal syntax for invariants here. It largely follows from the syntax of action guards, but is more powerful. The relaxed invariant $R(inv)$ results from relaxing all the conjuncts in the exact invariant inv. It is defined by the following rewrite rules in which we write $\mapsto$ for textual replacement:[4]

1	$R(q_1 \wedge \ldots \wedge q_g)$	$\mapsto$	$R(q_1) \wedge \ldots \wedge R(q_g)$
2	$R(\neg(r_1 \wedge \ldots \wedge r_h))$	$\mapsto$	$R(\neg r_1) \vee \ldots \vee R(\neg r_h)$
3	$R(\neg p(d.\vec{val}^{\backprime}, \vec{v}))$	$\mapsto$	$\neg p(d.\vec{val}^{\backprime}, \vec{v}) \vee \bigvee_{i=1}^{\ell} now - d_i.t^{\backprime} < \varepsilon_{d_i}$
4	$R(\neg(a(h.\vec{val}^{\backprime}, c.\vec{val}^{\backprime}, \vec{v}) \geq 0))$	$\mapsto$	$a(h.\vec{val}^{\backprime}, c.\vec{val}^{\backprime}, \vec{v}) < \varepsilon_a \vee \bigvee_{i=1}^{m} now - h_i.t^{\backprime} < \varepsilon_{h_i}$
5	$R(\neg(ts^{\backprime} \neq ts))$	$\mapsto$	$ts^{\backprime} = ts \vee now - ts^{\backprime} < \varepsilon_{ts}$

where the q_i are negated action guards, and the r_i are the atoms in the action guards. $x_i.t^{\backprime}$ refers to the current time stamp of discrete or hybrid channel x_i, $\vec{d}$ is an ℓ tuple, $\vec{h}$ is a m tuple, $d.\vec{val}^{\backprime}$, $h.\vec{val}^{\backprime}$, $c.\vec{val}^{\backprime}$, $\vec{v}$, $ts^{\backprime}$ and ts are as above, now is the HySChart's local clock, and all the ε_X are constants greater than 0. These constants have to be specified by the designer. The other identifiers are as above.

Before we explain the relaxation rules note that the central idea behind them is that a transition must be taken until the relaxed invariant becomes false. The relaxation ensures that some time passes between an action guard becoming true and the corresponding conjunct of the relaxed invariant becoming false. Furthermore, remember that action guards and invariants are checked in a HySChart at any point in time, because a HySChart's discrete part in the hybrid machine model permanently produces a next state (if it idles, this next state is equal to the latched state).

[4] Equality would be misleading, as described below.

The first two rules split the exact invariant into its negated atoms. The third rule covers propositions p over variables with domains different from the real numbers. Since the discrete topology is used for these variables, they are constant during anyone period of continuous evolution (cf. Section 3.5.2). Thus, they cannot violate the exact invariant $\neg p(d.\vec{val}', \vec{v})$ during a period of continuous evolution of the system, but only if a discrete jump occurs in one of the variables associated with discrete input channels. To allow the (relaxed) HySChart a delayed reaction to this jump, the disjunction in the relaxed invariant provides that the invariant remains true for some more time, until the jump is ε_{d_i} time units old w.r.t. the HySChart's local clock. In other words, when the disjunction is formulated in terms of an implication, the relaxed invariant is true as long as it holds that if the proposition p is true, then the last jump happened less than ε_{d_i} time units ago. Relaxing arithmetic constraints with rule 4 expresses that the relaxed invariant remains true for small (ε_a) overshooting over threshold 0. If the overshooting is caused by a discrete jump in a variable $h_i.val$ associated with a hybrid input channel, the invariant also remains true for some more time, until the jump is ε_{h_i} time units old. For time stamps (rule 5), the situation is similar to propositions since the discrete topology is used for them as well. Thus, they are also constant during anyone period of continuous evolution. Note that for the controlled variables $\vec{v}$, no time stamp information is used in the relaxation rules, since their possible discrete updates are controlled by the HySChart's discrete part *Com* and not by the environment, anyway. This means that if they are updated, a transition is already being taken and the old invariant no longer needs to hold. Moreover, for variables associated with continuous input channels no time stamp information in needed either, because they do not change discretely by definition.

Discussion. As a result of this relaxation of invariants, in a HySChart with relaxed invariants a discrete jump in the input need not immediately cause a transition to be taken, but the HySChart may remain idle for some more time. Similarly, threshold crossings by real valued, continuously changing input or controlled variables also cannot cause a transition to be taken immediately, but only when their value is significantly (by some ε_a) above the threshold. In both cases this ensures that when an action guard becomes true, the corresponding relaxed invariant still remains true for some time. Within this interval, the transition may be taken. One of the enabled transitions must be taken until the (relaxed) invariant becomes false.

To avoid that a discrete change, message or event is lost, i.e. that no reaction to it is enforced, each constant ε constraining the time tolerance w.r.t. a discrete change on a discrete or hybrid input channel must be chosen less than the minimum event separation on the respective channel. Remember that the minimum event separation is defined as the minimal time between two discrete changes on a channel (Section 3.4). For a relaxed invariant derived with rule 4, an important characteristic is that it remains true for values of the involved arithmetic expression which are within the permitted deviation ε_a above the threshold 0. Thus, the corresponding transition

need not be executed at all in this case (Figure 5.3). The ε_a reflects that boundary crossings cannot be detected with arbitrary precision.

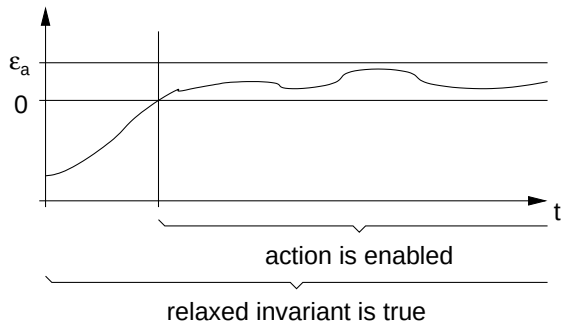


Figure 5.3: The relaxed invariant remains true for values up to the permitted deviation ε_a.

Action guards of the form $a = 0$ for an arithmetic expression a can be reduced to $-a \geq 0 \ \wedge \ a \geq 0$. Deriving the relaxed invariant for such a transition guard results in $-a < \varepsilon_1 \ \vee \ a < \varepsilon_2$ (plus additional disjuncts concerning hybrid channels). Obviously, this expression always is true for any $\varepsilon_1, \varepsilon_2 > 0$. It reflects that in presence of uncertainties it is not guaranteed that a transition which possibly only is enabled for single time instants is ever taken. In order to express that a transition is taken when an arithmetic expression a is almost zero, we therefore suggest to use macro $a =_\delta 0$ as action guard. It is defined by $a =_\delta 0 \equiv a \geq -\delta \ \wedge \ a \leq \delta$ for $\delta > 0$.

A further property of the relaxation method is that the relaxations of two semantically equivalent exact invariants need not be equivalent. In particular, the relaxation of an exact invariant which is unsatisfiable need not be unsatisfieable. As an example, consider the exact invariant $-a < 0 \ \wedge \ a < 0$ resulting from the action guards $a \geq 0$ and $a \leq 0$ for two transitions emerging from a given state. With the exact invariant, no time may pass in such a state since the predicate is always false. However, the relaxed invariant $-a < \varepsilon_1 \wedge a < \varepsilon_2$, for constants $\varepsilon_1, \varepsilon_2 > 0$, can be satisfied for values of a close to 0. Hence, some time may pass in the state with which the invariant is associated.

Finally, we point out that the relaxed discrete part which results from relaxing the invariants in the exact discrete part is guaranteed to be total if the exact discrete part is. This holds, because (1) whenever an input and a latched state satisfy an exact invariant they also satisfy the corresponding relaxed invariant and (2) additive hierarchic graphs are monotonous w.r.t. set inclusion (Section 3.3.5). Moreover, relaxed invariants derived in the described way define topologically open sets, because a relaxed invariant is given as a finite conjunction of disjunctions over predicates that define open sets (see above). Both these properties are required from the discrete part of a HySChart for well definedness of the hybrid computation model (Section 3.2.2).

Example 5.1 (Deriving a relaxed invariant.) The relaxed invariant of the state *inBend* of the EHC example (Figure 3.17, left) is derived as follows. The state has only one outgoing transition with action guard *bend*?, which stands for $bend.t` \neq bend.t$. Thus, the state's exact invariant is $\neg(bend.t` \neq bend.t)$. Rule 5 of the relaxation yields relaxed invariant $bend.t` = bend.t \vee now - bend.t` < \varepsilon_{bend}$ for a constant $\varepsilon_{bend} > 0$. We will encounter this invariant again in the example of Section 5.2.5 which also contains further relaxed invariants. □

5.2.2 Relaxed Analog Dynamics

Corresponding to the possible effects arising from sampling we define two kinds of relaxations for the analog dynamics. First, we relax the analog part by introducing a permitted deviation from the specified activities which is also visible to the discrete part. A second, additional relaxation of the continuous dynamics is introduced at the external interface of the HySChart to permit further relaxation, which does not affect the discrete part. In a discrete-time implementation of the analog part the first relaxation will comprise the discretization error while the second comprises the intersample error.

Syntax. On syntax level we associate each activity definition with a set of constants $\varepsilon_{dis}.x \geq 0$ for each controlled variable x occurring in the definition. Furthermore, the top-level hiararchic state (or *root node*) of a relaxed HySChart is annotated with an *output relaxation label* referring to a set of constants $\varepsilon_{int}.x \geq 0$ for each output variable x of the HySChart. The constants ε_{dis} define the deviation of allowed trajectories for the regarded controlled variable from the ideal trajectories as specified by the activity. This kind of deviation is visible to *Com* and it limits the allowed discretization error. The constants ε_{int} specify the output variables' further permitted deviation visible at the interface of the HySChart. This kind of deviations limits the permitted intersample error in a sampling implementation. By convention deviations ε_{dis} are only specified for real valued controlled variables excluding time stamps and similarly deviations ε_{int} are only specified for real valued output variables excluding time stamps. The other variables are supposed to be unaffected by activities anyway. For variables x for which no deviation is explicitly given, we implicitly use $\varepsilon_{dis}.x = 0$ and $\varepsilon_{int}.x = 0$. In particular, this provides that $\varepsilon_{int}.x$ is 0 for those variables not referenced in an activity's definition. Furthermore, for variable *now*, $\varepsilon_{dis}.now$ is required to be 0, since wrong time values in HySCharts may corrupt the detection of messages and jumps in the input.

Relaxed analog part. Semantically the relaxed analog part *RAna* results from the relaxation of the primitive activities. The relaxation $R(A)$ of a primitive activity A is defined as consisting of all tuples of smooth trajectories $((\iota, \tau), \tau)$ such that τ is not more than the given deviation away from a trajectory σ which satisfies activity A for

input ι. We say that a state-stream σ *satisfies* an activity A for input stream ι iff $((\iota,\sigma),\sigma) \in A$. Formally for $A \subseteq (\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}}) \times \mathcal{S}^{\mathbb{R}_{s+}}$, $R(A)$ is defined by:

$$\begin{aligned} R(A) \;=\; & \textstyle\bigcup_{((\iota,\sigma),\sigma)\in A}\{((\iota,\tau),\tau) \in (\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}}) \times \mathcal{S}^{\mathbb{R}_{s+}} \mid \\ & \forall x \in CV.\, \forall t \in \mathbb{R}_+ .\, d_x(\sigma.x(t), \tau.x(t)) \leq \varepsilon_{dis}(A).x\} \end{aligned}$$

where CV is the set of controlled variables of the HySChart, d_x is a metric on the domain of controlled variable x, the $\varepsilon_{dis}(A).x$ are the permitted deviations which are associated with activity A as described above, and $\sigma.x$ and $\tau.x$ denotes the evolution of variable x in the respective stream. For real valued controlled variables excluding time stamps we use the natural metric on the real line and for variables with other domains and time stamps we use the discrete metric. Together with the above conventions this provides that variables with domains different from the real numbers and time stamps are not modified by the relaxation. Similarly, the variables not occurring in an activity's definition also remain unchanged due to the conventions above. Like the original activity, the relaxed activity also only contains smooth trajectories, because of its type. Resolvability of activities, which helps to ensure well definedness of the hybrid machine model, is maintained by the relaxation, because the exact activity is a subset of the relaxed one. Furthermore, the relaxation is defined such that the received state-stream and the produced state-stream coincide for a relaxed activity, just as for exact activities. As explained in Section 3.5.2 this ensures that the discrete and the analog part in a hybrid machine agree on the same smooth evolution for the state. Thus, the interaction of the discrete part and the relaxed analog part is just as in a HySChart without relaxation.

According to Section 3.5.2 the exact analog part *Ana* is given as a disjoint sum of sequentially composed, adapted activities. With the above base case the relaxation of *Ana* can therefore be defined inductively by the following rewrite rules, in which $\mapsto$ denotes textual replacement:

$$\begin{aligned} R(A{+}B) &\mapsto R(A){+}R(B) \\ R(A\,;_\times\, B) &\mapsto R(A)\,;_\times\, R(B) \\ R(da(A)) &\mapsto da(R(A)) \end{aligned}$$

The relaxed analog part *RAna* is defined by $RAna = R(Ana)$. Like the exact analog part, it is resolvable if all (relaxed) activities are (cf. Section 3.5.2). Note that we above demanded that the allowed deviation for variable *now* is 0. Provided *now* is zero initially, this yields that *now* always contains the correct time since system start.

Output Relaxation. Further relaxation of the analog dynamics results from including a relaxation component R_{int} at a HySChart's output interface. This component is defined as follows:

$$\begin{aligned} R_{int} \;&\in\; \mathcal{O}^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_{p+}}) \\ R_{int}(\sigma) \;&=\; \{\tau \mid \forall x \in OV.\, \forall t \in \mathbb{R}_+ .\, d_x(\sigma.x(t), \tau.x(t)) \leq \varepsilon_{int}.x\} \end{aligned}$$

where OV denotes the set of output variables of the HySChart, $\mathcal{O}$ is the associated output space, $\varepsilon_{int}.x$ is the permitted intersample deviation for output variable x, as

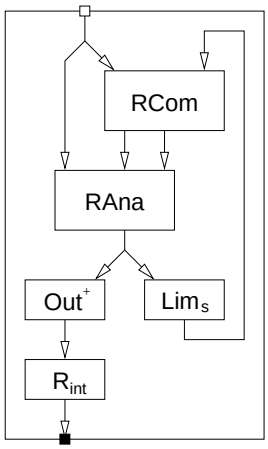


Figure 5.4: Machine model for relaxed HySCharts.

specified for the HySChart, and d_x, $\sigma.x$ and $\tau.x$ are as in the previous paragraph. In contrast to the relaxation of activities, R_{int} can add discontinuities to the trajectories it receives. They can jump arbitrarily in the neighborhood of the received stream σ as long as they lie within $\varepsilon_{int}.x$ distance from $\sigma.x$ for every output variable x and as long as they remain piecewise smooth and piecewise Lipschitz continuous. This is enforced by the type of R_{int}. To simplify notation in the proofs in Appendix A.2.2 we overload R_{int} to finite streams $\sigma \in (n \cdot \mathcal{S})^{[0,t)}$ for $t > 0$ and write $R_{int}(\sigma)$ instead of $(R_{int}|_{[0,t)})(\sigma)$.

5.2.3 Machine Model for Relaxed HySCharts

The machine model for relaxed HySCharts results from using the relaxed discrete part which results from relaxing the invariants and which we denote by $RCom$, the relaxed analog part $RAna$ and from adding output relaxation R_{int} at the output interface of the HySChart. Figure 5.4 depicts the machine model. The corresponding formula is:

$$((\aleph_2^{\mathcal{I}} \times \mathsf{I}_{n\cdot\mathcal{S}}) \;;_\times\; (\mathsf{I}_{\mathcal{I}} \times RCom^\dagger) \;;_\times\; RAna \;;_\times\; \aleph_2^{n\cdot\mathcal{S}} ;_\times\; (Out^\dagger \times Lim_s)) \uparrow_\times^{n\cdot\mathcal{S}} \;;_\times\; R_{int}$$

where $\mathcal{I}$ and $n \cdot \mathcal{S}$ denote the input space and the state space of the machine, respectively. The state-based semantics results from replacing the time extended output projection $Out^\dagger$ and the output relaxation R_{int} in the formula by the identity $\mathsf{I}_{n\cdot\mathcal{S}}$.

From a mathematical point of view the relaxed discrete part and the relaxed analog part are special cases of the discrete and analog part as defined in Chapter 3. They have the same interface and satisfy the same smoothness restrictions. Hence, besides R_{int}, relaxed HySCharts are ordinary HySCharts with a few special characteristics. As outlined in Section 5.2.1, $RCom$ is total and its invariants are open sets if the same holds for the exact discrete part. Moreover, $RAna$ is resolvable if the activities in the exact analog part are (Section 5.2.2). Therefore, well definedness of (exact)

HySCharts carries over to relaxed HySCharts and the inductive proof principle introduced in Appendix A.1.2 can be used similarly for the state-based semantics of relaxed HySCharts. Due to this correspondence, we also write St to denote the state-based semantics of relaxed HySCharts, just as for exact HySCharts. The state-based semantics will be used in proofs involving relaxed HySCharts, because it allows us to reason about the machine's internal state.

If we regard the output values of *RAna* as approximations to exact values, as motivated before, it is important to consider the propagation of the error resulting from the approximation. By definition of the relaxation of the activities the error remains bounded during periods of continuous evolution. It can, however, accumulate over transitions by the discrete part, if the corresponding action bodies do not reinitialize the start values of those controlled variables which are affected by the error. The reason is that the variables' values after a transition are interpreted as exact start values for the next period of smooth evolution although they possibly contain errors. Apart from that, the error introduced by *RAna* relaxes the constraints given in the relaxed discrete part *RCom* further. For instance, a relaxed invariant of the form $x < c + \varepsilon$ means that the invariant becomes false when the x resulting from the relaxed analog part does not satisfy the inequation. With respect to the exact value of x, denoted by $\overline{x}$, this in the worst case means that the relaxed invariant may remain true until $\overline{x} = c + \varepsilon + \varepsilon_{dis}.x$. Thus, the relaxation by *RCom* is increased further by ε_{dis}. Similarly, the analog part and the discrete part are also affected by errors in their external input.

Besides that, the Lipschitz constants for the evolutions of the controlled variables are affected by the relaxation, in particular by R_{int}.[5] Assuming that the evolution of variable x as specified by an activity A is Lipschitz continuous with Lipschitz constant l, the evolution of the trajectories in the relaxed activity is constrained as follows. Let σ be the exact evolution of x and let τ be the evolution according to the relaxed activity. We get that the maximal distance between the relaxed values of x at two time points t_1 and t_2 is limited by $d_x(\tau(t_1), \tau(t_2)) \leq d_x(\sigma(t_1), \sigma(t_2)) + d_x(\tau(t_1), \sigma(t_1)) + d_x(\tau(t_2), \sigma(t_2)) \leq l \cdot |t_1 - t_2| + 2\varepsilon_{dis}.x$. While this estimate does not yield the Lipschitz constant of τ it nevertheless permits to estimate possible values of $\tau(t_2)$ at given t_2 when a value $\tau(t_1)$ is given. The effect of the relaxation at the HySChart's output interface during an interval δ where the output of *RAna* is smooth and Lipschitz continuous can be estimated similarly. Let ν be the evolution at the output interface of the relaxed HySChart and let $t_1, t_2 \in \delta$. Then we get $d_x(\nu(t_1), \nu(t_2)) \leq l \cdot |t_1 - t_2| + 2(\varepsilon_{dis}.x + \varepsilon_{int}.x)$. Note that the relaxed trajectory ν need not be continuous on δ. Nevertheless, its value at time t_2 is constrained by l, ε_{dis}, ε_{int} and $\nu(t_1)$. As we will see in Section 5.4.2, such constraints can be used to select sampling rates for the discrete-time implementation of a HySChart.

[5] The type of relaxed activities and of the output relaxation R_{int} ensures that they indeed evolve Lipschitz continuously.

5.2.4 Remarks on the Relaxation

The relaxation of the discrete and the analog part may cause behavior that is qualitatively different from the behavior of the exact HySChart. For instance, the relaxation may result in variable values that make the guard of a transition true which would never be enabled in the exact HySChart. The new control state, which becomes reachable this way, may lead to entirely different further evolution. In contrast, the relaxation by ε_{int} at the HySChart's output interface only introduces further behavior that is quantitatively different, but qualitatively similar to the behavior without this relaxation. By definition of R_{int} it only introduces deviations in the values of output variables. As these deviations are not visible to the discrete part, but only added at the HySChart's interface, they cannot trigger transitions in the HySChart and create qualitatively new behavior. Nevertheless, the deviations can affect other external components. In Section 6.5.1 we verify two properties of a relaxed HySChart for our example system, the EHC. The feasibility of this verification demonstrates that if a model is designed properly and reasonable constants are used in the relaxation, vital properties hold for the relaxed system and can be proven.

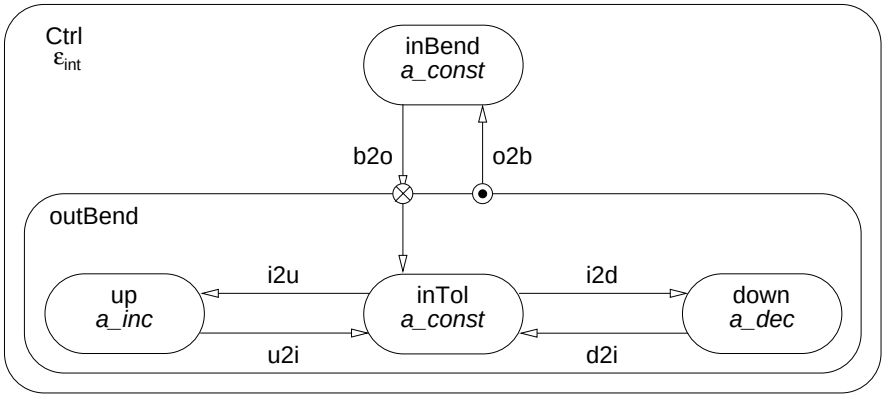
In cases where activities are not given as (detailed) differential equations, but as liberal differential constraints, relaxation of the analog part is not needed to permit flexible refinement in later development steps. Typically it is needed for refinement if an activity is given as an initial value problem with a unique solution, in particular if this solution is difficult to compute numerically. Output relaxation is nevertheless required to allow for implementation effects which result at sampling instants when the discrete-time output is simply held constant between sampling instants.

The deviations we introduced reflect pessimistic assumptions about effects possibly occurring in implementations. A design that is robust against such kinds of deviations must be able to tolerate them. Designs which are only correct in presence of irrealistic assumptions, like infinitely fast response, will not work in the real world. Therefore, we think that demanding that a relaxed HySChart satisfies some desired property is not a too strong proof obligation, but reasonable, since it enables an implementation of the design which also satisfies the desired property.

Finally, note that the relaxed HySChart is an *abstraction* of the exact one, or dually, the exact one is a refinement of the relaxed HySChart. This is straightforward by setting all the constants ε used in the relaxation to 0.

5.2.5 Relaxed HySChart for the EHC's Controller

Here we explain the relaxed HySChart for the EHC's controller, and outline how it is obtained from action guards, activities and a set of constants with the methods from above. At the stage of the development process we consider, the overall system



Actions:

$b2o$	$\equiv$	$bend?$
$o2b$	$\equiv$	$b2o$
$i2u$	$\equiv$	$fHeight.val` \leq lb$
$i2d$	$\equiv$	$fHeight.val` \geq ub$
$u2i$	$\equiv$	$fHeight.val` \geq lb + c \ \wedge \ reset!$
$d2i$	$\equiv$	$fHeight.val` \leq ub - c \ \wedge \ reset!$

Relaxed invariants (derived):

$Ctrl_{inv}$	$\equiv$	$True$
$inBend_{inv}$	$\equiv$	$bend.t = bend.t` \ \vee \ now - bend.t` < \varepsilon_{bend}$
$outBend_{inv}$	$\equiv$	$inBend_{inv}$
$inTol_{inv}$	$\equiv$	$fHeight.val` \in (lb - \varepsilon_{i1}, ub + \varepsilon_{i2}) \ \vee \ now - fHeight.t` < \varepsilon_{fHeight}$
up_{inv}	$\equiv$	$fHeight.val` < lb + c + \varepsilon_u \ \vee \ now - fHeight.t` < \varepsilon_{fHeight}$
$down_{inv}$	$\equiv$	$fHeight.val` > ub - c - \varepsilon_d \ \vee \ now - fHeight.t` < \varepsilon_{fHeight}$

(Relaxed) activities:

a_const	$\equiv$	$\frac{d}{dt} aHeight = 0$ with $\varepsilon_{dis}.aHeight = 0$
a_inc	$\equiv$	$\frac{d}{dt} aHeight \in [cp_-, cp_+]$ with $\varepsilon_{dis}.aHeight = 0$
a_dec	$\equiv$	$\frac{d}{dt} aHeight \in [ev_-, ev_+]$ with $\varepsilon_{dis}.aHeight = 0$

Output relaxation constants: $\varepsilon_{int}.aHeight$

Figure 5.5: (Relaxed) HySChart for the abstract controller.

has already been split into the components *Environment, Filter, Controller* and *Delay element* as depicted in Figure 3.2, left.

Example 5.2 The controller is given by the HySChart in Figure 5.5 which also lists the actions, (relaxed) invariants and (relaxed) activities. Its input interface consists of channels *bend* and *fHeight*, and its output interface contains the channels *aHeight* and *reset*. The informal meaning of the states is as in the preceding chapter. The actions $b2o$ and $o2b$ model the controller's reaction to events on channel *bend*, i.e. to entry and exit of bends. The other actions only depend on the filtered chassis level

as provided by the *Filter* component and determine whether the chassis level needs to be increased, decreased or left unmodified by the controller (activities *a_inc*, *a_dec* and *a_const*). The action bodies of $u2i$ and $d2i$ send the *reset* signal to the filter. Constant $c > 0$ in the action guards of $u2i$ and $d2i$ ensures that once an actuator (escape valve or compressor in the physical system) is activated, it is used to drive the filtered chassis level into the tolerance interval $[lb, ub]$, not just to its boundary. The constant is chosen such that $lb + c < ub - c$.

The relaxed invariants listed in Figure 5.5 are derived from the action guards in the way described in Section 5.2.1. Note that we simplied the resulting predicates to increase clarity of the presentation. For instance, the relaxed invariant of *inTol* without simplification is:

$$(-fHeight.val` + lb < \varepsilon_{i1} \ \vee \ now - fHeight.t` < \varepsilon_{fHeight}) \ \wedge$$
$$(fHeight.val` - ub < \varepsilon_{i2} \ \vee \ now - fHeight.t` < \varepsilon_{fHeight})$$

Apart from the simplification, the relaxed invariants can be constructed automatically by a CASE tool. The designer only has to specify transition guards and the non-negative constants $\varepsilon_{bend}, \varepsilon_{fHeight}, \varepsilon_{i1}, \varepsilon_{i2}, \varepsilon_u$ and ε_d used in the relaxation. The first two of these constants determine how long reaction to a *bend* event or a discrete jump in *fHeight* may be delayed. The others constrain the permitted overshoot of *fHeight* over the thresholds given in the transition guards. Here, we set $\varepsilon_{i1} = \varepsilon_{i2} = \varepsilon_u = \varepsilon_d$ and select the constant such that $lb < ub - c - \varepsilon_d$ and $lb + c + \varepsilon_u < ub$. This provides that it is never possible to go from state *up* to *down*, or vice versa, without remaining in state *inTol* for some time. Such a behavior would be unreasonable for the EHC.

For the activities *a_inc*, *a_dec* and *a_const* the constants $\varepsilon_{dis}.aHeight$ of each activity are set to zero, because *a_inc* and *a_dec* already constrain the evolution of *aHeight* in a very liberal way. Although *a_const* is much stricter it is not relaxed either, because the requirement it represents is not difficult to satisfy in implementations. There, it suffices to switch compressor and escape valve off in order to achieve the behavior required by activity *a_const*. The relaxed analog part therefore is identical to the analog part without relaxation. The output relaxation R_{int} of *aHeight* at the component's output interface if given by the positive constant $\varepsilon_{int}.aHeight$. As for the other constants, we do not give a concrete value here.

By convention the activity $\frac{d}{dt}now = 1$ of the root node *Ctrl* need not be written explicitly and its relaxation $\varepsilon_{dis}.now$ by convention is 0 (Sections 3.5.2 and 5.2.2). Variable *now* is not visible outside. Thus, no output relaxation is given for it. Furthermore, as variable *reset* is a time stamp, it is not relaxed either, i.e. the relaxation constant ε_{reset} is implicitly set to zero.

While we do not prove any properties of this relaxed HySChart we want to point out that this model can be regarded as a refinement of the more abstract relaxed model in Section 6.5.1. There, two vital properties for the abstract model are established. □

5.3 DiSCharts

To specify a discrete-time implementation of a model which is given as a HySChart we need a description technique and an execution model for such an implementation. Therefore, we define *DiSCharts* in this section. Like HySCharts, DiSCharts are hierarchic, sequential control-flow graphs used for the specification of the behavior of system components, but in contrast to HySCharts their underlying time model is discrete. We use the natural numbers. Besides that, they are defined very similar to HySCharts, based on the hierarchic graph framework introduced in Section 3.3. This enables us to define a simple methodology that supports the transition from a relaxed HySChart to a DiSChart in Section 5.4. The guiding principle in the definition of DiSCharts is to keep them close to HySCharts. The syntax of DiSCharts is the same as for HySCharts (cf. Section 3.5), only the (discrete-time) activities in DiSCharts refer to another kind of predicates. In the spirit of the integrated development process of Section 2.2.2, a model specified as a DiSChart need not already define the final implementation. Instead it may be the basis for further development steps in discrete time. This motivates that we define DiSCharts in a way which explicitly allows us nondeterministic specifications.

5.3.1 Machine Model

Semantically DiSCharts are time-guarded relations between discrete-time input and output communication histories, parameterized with a start state s_0. A discrete-time communication history (or discrete-time stream) is a mapping from the nonnegative natural numbers $\mathbb{N}$ to some domain A. The set of discrete-time streams with domain A is denoted by $A^{\mathbb{N}}$. Thus, the type of a DiSChart is $n \cdot \mathcal{S} \to (\mathcal{I}^{\mathbb{N}} \to \mathcal{P}(\mathcal{O}^{\mathbb{N}}))$, where $n \cdot \mathcal{S}$ is the state-space of the DiSChart, $\mathcal{I}$ its input space and $\mathcal{O}$ its output space, just as for HySCharts. Like before, the state space consists of the control-state space and the data-state space, which defines the values of all controlled variables.

The semantics of DiSCharts is defined based on the machine model depicted in Figure 5.6.[6] It basically is a Moore machine. *DAna* and *DCom* are discrete-time versions of the analog part and the discrete part as known from HySCharts. *DAna* is an algorithm which at every time instant computes the value of the controlled variables according to some mathematical evolution law, for instance a difference equation or a numerical integration algorithm. *DCom* is a state transition relation which uses the state provided by *DAna* and the external input to determine the next state. Δ_s is a discrete delay by one time unit. At any time point k its output is the input it received at the previous time point or the initial state, if $k = 0$. Formally

[6] Formally, the figure depicts a multiplicative hierarchic graph which is interpreted w.r.t. time model $\mathbb{N}$.

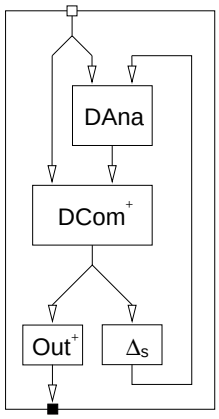


Figure 5.6: Machine model for DiSCharts.

$\Delta_s(\sigma)(k) = \sigma(k-1)$ if $k > 0$ and s otherwise. Like in HySCharts, the state provided as output of Δ_s is called the *latched state*.

At each time instant *DAna* receives the latched state and the current external input. Under the assumption that time T passes between two time instants (or *activations*), it updates the latched state according to the evolution laws encoded in it. The updated state and the current input is used by *DCom* to decide whether it idles or produces a new next state different from the updated state. Whether new or not, the next state is given to the output interface of the DiSChart and to Δ_s where it is stored until the next activation of the chart. The projection *Out* makes the values of the output variables visible outside.

In comparison to the HySChart machine model (see Figure 3.4, middle) the succession of *DAna* and *DCom* is reversed in the DiSChart. This is motivated by the desire to keep the effects of the delay Δ_s small. At a time point k, Δ_s gives the latched state to *DAna* and *DAna* can compute an approximation of the controlled variables corresponding to the present external input and assumed physical time $k \cdot T$, where T again is the sampling rate with which the DiSChart is supposed to operate. This approximation can immediately be used by *DCom* to decide whether it produces a new next state or idles. If the succession was like in the HySChart machine model, i.e. *DCom* would preceede *DAna*, *DCom* would have to base this decision on the latched state which was computed by *DAna* for a different physical time and old external input. As *DAna* is responsible for correct values of the components local clock *now*, this would e.g. mean that *DCom* has to work with old clock values. Wrong clock values are critical, since the detection and sending of events relies on time stamps, like in HySCharts. Furthermore, old values of *DAna* can cause completely inappropriate reaction of *DCom* to its current input.[7] On the other hand, no time is wasted if *DAna*

[7] In control theory, delays of this kind are known to be able to cause *instability* of a system (Section 6.3.3).

cannot react to a discrete move by *DCom* at the same time instant, since, like in a HySChart, we require that the DiSChart's output right after a transition is the state produced by *DCom*, not a modified version of it. Thus, if *DAna* followed after *DCom* in the machine model, it would not be allowed to update the state it receives in such a situation, anyway.

5.3.2 The Discrete Part in Discrete Time

The discrete part of a DiSChart *DCom* is essentially equivalent to the discrete part in HySCharts. It is a relation without time. At each activation it maps the current input and the updated state to the next state, formally: $DCom \in (\mathcal{I} \times n \cdot \mathcal{S}) \to \mathcal{P}(n \cdot \mathcal{S})$, where $n \cdot \mathcal{S}$ is the program-state space of the DiSChart. $DCom(i, s)$ must be nonempty for any $(i, s) \in \mathcal{I} \times n \cdot \mathcal{S}$, i.e. the relation is total in $\mathcal{I} \times n \cdot \mathcal{S}$. It is derived from a hierarchic graph in the same way as described in Section 3.5.1 for the discrete part of HySCharts. The syntax of guards, actions and invariants as well as the encoding of events and messages is as defined there. For DiSCharts, however, invariants need not define topologically open sets. We can relax this condition, because the finitely large delay Δ_s in the machine model guarantees its well definedness (see below). Like HySCharts, DiSCharts also contain a controlled variable *now* which contains the current physical time at each time point when *DCom* is activated. Variable *now* is updated by *DAna* under the assumption that the physical time between two successive activations of the DiSChart is T. It is needed to set the time stamps for event and message based communication. Additionally, DiSCharts contain a further special variable $rs \in \{init, true, false\}$, denoting a kind of reset. It is used by the discrete part to signal *DAna* when it has taken a transition and also for initialization of the DiSChart. In the discrete-time implementation *DAna* can no longer detect discontinuities caused by transitions in *DCom*. Therefore, *DCom* explicitly has to signal when a transition is taken such that *DAna* can react by changing the present activity. From an implementation point of view such a change can be regarded as a reinitialization of the used integration algorithms. We implicitly associate action body $rs' = true$ with every transition in the DiSChart. If no transition is taken, *DCom* leaves *rs* unchanged.

5.3.3 The Analog Part in Discrete Time

As in HySCharts, a DiSChart's behavior is determined by activities during the time periods where the discrete part idles. An activitiy in discrete-time is a relation *DAct* with type $DAct \subseteq (\mathcal{I}^{\mathbb{N}} \times \mathcal{S}^{\mathbb{N}}) \times \mathcal{S}^{\mathbb{N}}$. Activities must be time-guarded. We also require that activities are independent from absolute timing, like in the continuous time case.

Activity definition. In discrete-time a tuple of streams $((\iota, \sigma), \tau)$ in an activity is regarded as resulting from one of a set of *computation laws* (or algorithms) which define the activity. A computation law is a relation of the same type as actions in the

discrete part, i.e. a relation between the current input, the latched state and the next state. It must be total in the input and the latched state. The computation law is applied at each time instant to yield state stream τ from input stream ι and latched state stream σ. Thus, for a set of computation laws CL with $cl \subseteq \mathcal{I} \times \mathcal{S} \times \mathcal{S}$ for any $cl \in CL$, the discrete-time activity $DAct$ is defined by:

$$DAct = \bigcup_{cl \in CL} cl^{\dagger}$$

where $cl^{\dagger}$ is the time extension of cl w.r.t. time model $\mathbb{N}$. Note that the selection of cl cannot be changed as time evolves.

In comparison to continuous-time, computation laws can be regarded as resulting from difference equations, similar to differential equations which specify continuous-time activities. The intuition is that computation laws compute an approximation for some continuous evolution based on the assumption that physical time T passes between two successive activations of a DiSChart. Numerical integration algorithms are typical candidates for computation laws. As described in more detail in Section 5.4.3.2, such algorithms sometimes treat initialization in an exceptional manner and some algorithms also need to store past values. For the purpose of initialization we included variable rs into our data-state space. Its usage is explained below. Past values can also be stored as part of the data state. As such values are irrelevant for the discrete part and only needed by $DAna$ for its computations, we refer to the part of the data-state space which is only needed for auxiliary variables of $DAna$ as $\mathcal{S}_{aux}$ and write $\mathcal{S}_{re}$ for the rest of the data-state space. We use a set of computation laws instead of a single computation law as specification for an activity, because this allows us to specify different kinds of evolutions for the state. These evolutions can be regarded as corresponding to different approximated solutions of an initial value problem, or to different approximation methods. In later development steps a particular computation law can be selected. For instance, this can be useful to select a particular discretized linear evolution for a variable from a set of such evolutions. This is considered in the example in Section 5.3.6.

Like actions, we describe computation laws by their characteristic predicate and use the same conventions as for actions to refer to current inputs, current and latched time stamps, and next values of controlled variables. For instance, the computation law updating variable now with the current time provided $rs \neq init$ is written as:

$$(rs \neq init \Rightarrow now' = now + T \ \wedge \ rs' = rs) \ \wedge \ (rs = init \Rightarrow now' = now \ \wedge \ rs' = rs)$$

If $rs = init$ the value of now remains unchanged. The value of rs remains unchanged in both cases. For activities which are associated with leaf nodes of a DiSChart, this is defined differently below.[8] Note that this updating of now corresponds to the

[8]Like for HySCharts, leaf nodes are those nodes in the hierarchic graph for a DiSChart which have no subnodes.

difference equation $\frac{now'-now}{T} = 1$ where the physical time passing between two time instants in the discrete-time model is assumed to be T. By convention we usually only explicitly write the part of the computation law which defines the behavior in case of $rs \neq init$ and do not explicitly specify how rs is modified by the computation law, since this is determined by the conventions given below. Furthermore, we omit equalities $v' = v$ stating that the value of a variable v does not change in a computation law. Thus, variables v whose next value v' does not appear in the predicate remain unchanged. For the computation law from above we can therefore use the shorthand $now' = now + T$, where the treatment of rs is implicit. In a tool environment, the implicit part of the predicates can be added by the tool.

Composition of activities. Similar to HySCharts, the discrete-time analog part is obtained by pasting traces of the activities associated with the control states where *DCom* idles. Pasting is realized by first adapting activities such that they may be reset in reaction to moves by *DCom* and then by switching between the adapted activities with the time-extended disjoint sum (Section 3.3.4), similar to the continuous-time case. The adaptation to resets is defined by overloading the discontinuity adaptation $da(.)$ to discrete-time activities in the following way.

$$\begin{aligned} da(DAct) \;=\; \{\; &((\iota,\sigma),\tau) \in (\mathcal{I}^{\mathbb{N}} \times \mathcal{S}^{\mathbb{N}}) \times \mathcal{S}^{\mathbb{N}} \mid \\ &\forall \delta \in Int \cap \mathbb{N}.\, \sigma.rs|_\delta = \rho_\delta \Rightarrow ((\iota,\sigma),\tau)|_\delta \in DAct|_\delta\} \end{aligned}$$

where Int is the set of all intervals over $\mathbb{R}_+$ and ρ_δ is a function on δ defined by $\rho(min(\delta)) \in \{init, true\}$ and $\rho(k) = false$ for all $k \neq min(\delta)$. The definition expresses that the computation law selected from an activity may be changed whether rs is *true* or *init*. Then, the selection remains constant until the next reset, caused by a transition in *DCom*. Adaption $da(.)$ would not be needed, if *DAct* was given by a single computation law. Like in the continuous-time case, $da(.)$ distributes over the sequential composition of activities.

As in HySCharts, the nodes in the hierarchic graph for a DiSChart are labeled with activity names, which refer to activities. The discrete-time analog part is derived from the hierarchic graph as described in Section 3.5.2 for the analog part of HySCharts. Thus, it also is a disjoint sum of sequentially composed, adapted activities. This means that it can be written in the form $DAna = +_{i=1}^{n} (;_{+j=1}^{m_i} da(DAct_{i,j}))$, where the underlying time model for the operators here is $\mathbb{N}$. As activities result from the time extension of computation laws, which are required to be total, the analog part is time guarded and total in the received input and state stream.

Conventions. If no activity is specified for a node in a DiSChart, the time extendended additive identity relation $\mathrm{i}_{\mathcal{S}} = \{((\iota,\sigma),\sigma) \mid \iota \in \mathcal{I}^{\mathbb{N}} \wedge \sigma \in \mathcal{S}^{\mathbb{N}}\}$ is used as activity semantics for that node. Thus, the state is not changed here.

As far as initialization is concerned, we require that if $rs = init$, which signals that this is the first activation, all activities associated with hierarchic nodes in the DiSChart at most modify the part of the data state which is only needed for auxiliary variables

of *DAna*, denoted by $\mathcal{S}_{aux}$. For $rs = init$, activities associated with primitive nodes must set rs to *true* and also leave the rest of $\mathcal{S}_{re}$ unchanged. This is required, because at system start ("time 0") the DiSChart and its discrete part are supposed to start working with the specified initial state s and not with an approximation for the state at the next time instant ("time 1") which would be computed by the first activation of an activity's computation law. Nevertheless, activities can use this startup phase for initializing those parts of the data-state space $\mathcal{S}_{aux}$ which are exclusively needed by activities, e.g. to remember old inputs. Setting rs to *true* when the initialization is completed allows the analog part to start a new activity at the time instant after initialization.[9]

Furthermore, we require that each activity which is associated with a primitive node in the DiSChart sets rs to *false*, if its latched value is different from *init*. This ensures that the (re)start of activities caused by $rs = true$ ends when *DAna* passes the program state to *DCom*. Activities associated with hierarchic nodes do not modify rs, because the original value is still needed by the (sequentially composed) activities of its subnodes. Together with *DCom* setting rs to *true* iff it takes a transition, this ensures that the control state does not change during the intervals in which rs is *init* or *true* initially and *false* thereafter. This in particular motivates the definition of $da(.)$ above. In examples, the modification of rs usually is not specified explicitly since it follows from these conventions.

By convention we implicitly add a predicate that updates now to each computation law of the primitive activity which is associated with the top-level hierarchic node (root node) of the DiSChart. The predicate was already given above, omitting the implicit treatment of rs it is $now' = now + T$. Provided now is initialized with zero, $rs = init$ initially, and the activation rate T is correct, the resulting activity within the machine model of Figure 5.6 ensures that now denotes the time since system start.

Remarks. The principal difference between the discrete part and the analog part in a HySChart, which consists in the analog part being responsible for the continuous dynamics while the discrete part is responsible for discontinuities, vanishes here. As the discrete-time output of *DAna* is only given at isolated time instants $k \in \mathbb{N}$, the notion of continuous evolution is no longer meaningful. From a methodical point of view, however, the algorithms in *DAna* are expected to specify some kind of regular evolution of the controlled variables over time. To denote such a regular evolution, the computation laws defining activities could e.g. be specified by (simple) discrete-time block diagrams which represent difference equations. The methodical intention behind *DCom* is that it is responsible for logical decision making while *DAna* does not make decisions, besides the treatment of the rs signal.

When a DiSChart reacts to the sampled values on a hybrid input channel with its

[9] Remember that the last activity in a sequential composition of activities in the analog part stems from a primitive node. In other words, by construction of the analog part the sequential compositions of activities in it reflect the hierarchy in the DiSChart (cf. Section 3.5.2).

underlying discrete time grid, jumps in the values on the channel cannot be detected by the analog part and *DAna* therefore cannot react to them by choosing a new computation law from the current activity. If such a change as reaction to jumps in the input is desired, *DCom* must observe the time stamp information associated with the input channel and initiate a change by taking a transition, thereby setting *rs* to *true*.

5.3.4 The Discrete-Time Component

The denotational semantics *DCmp* of a component specified by a DiSChart is given by writing the graph in Figure 5.6 as a relational expression with the multiplicative operators for underlying time model $\mathbb{N}$:

$$DCmp(s) \;\; = ((\wp_2^{\mathcal{I}} \times \mathsf{I}_{n\cdot\mathcal{S}}) \;;_\times\; (\mathsf{I}_{\mathcal{I}} \times DAna) \;;_\times\; DCom^\dagger \;;_\times\; \wp_2^{n\cdot\mathcal{S}} \;;_\times\; (Out^\dagger \times \Delta_s)) \uparrow_\times^{n\cdot\mathcal{S}}$$

where $\mathcal{I}$ is the input space and $n \cdot \mathcal{S}$ is the program-state space, as usual. Provided *DCom* is total and *DAna* is total and time guarded, *DCmp* also is total and time guarded. Unlike for HySCharts, this result immediately follows from the finitely large delay Δ_s which is contained in the feedback loop [GR95]. If totality holds, the type of $DCmp(s)$ is $\mathcal{I}^{\mathbb{N}} \to \mathcal{P}(\mathcal{O}^{\mathbb{N}})$, where $\mathcal{O}$ is the output space. For the "white-box" (or state-based) semantics of a component we write *DSt*. It is defined by replacing $Out^\dagger$ by the identity $\mathsf{I}_{n\cdot\mathcal{S}}$ in the above formula for *DCmp*. We need it to reason about the DiSChart's internal behavior.

Similar to HySCharts, the definition of the analog part as resulting from pasting the output of the activities whenever *DCom* idles results in an operational semantics for DiSCharts. A step in this semantics consists of a discrete move by *DCom* followed by a period of evolution where the behavior is determined by *DAna* and *DCom* idles. This semantics is given in Appendix A.2.1 and used to proof the correctness of the refinement principle given in the next section.

Note that during time periods where *DCom* idles, the state stream produced by *DAna* is fed back to it with a delay of one time unit. We say that stream $\tau \in (n \cdot \mathcal{S})^{\{k_1,\ldots,k_2+1\}}$ *satisfies RAna for input* $\iota \in \mathcal{I}^{\{k_1+1,\ldots,k_2+1\}}$ iff it complies with this feedback composition. Formally, this is expressed as $(\iota, \tau^1|_\delta, \tau|_\delta) \in DAna|_\delta$, where $\delta = \{k_1 + 1, \ldots, k_2 + 1\}$, $k_1 \leq k_2$ in $\mathbb{N}$, and τ^1 is the right shift of τ by one time unit. The right shift expresses the effect of the delay. In the continuous-time case, the interaction of *Com* and *Ana* is similar. There, the produced state-stream of *Ana* is *directly* fed back to it when *Com* idles, because during such an idle period the stream is continuous, *Lim* therefore is the identity, and there is no further delay in the feedback loop (Section 3.2.4).

5.3.5 Interface to Dense Streams

As our aim is to use DiSCharts to specify those components of a hybrid system which are supposed to be implemented in software or digital hardware, we have to define how DiSCharts interact with the other components whose behavior is given in terms of dense streams. For the mapping from dense streams to discrete-time streams we use a sampling component $sample_{\mathcal{I},T}$ and for the reverse mapping we use a zero-order hold $hold_{\mathcal{O},T}$. Component $sample_{\mathcal{I},T}$ provides the value of its dense input stream at time instant $k \cdot T$, $k \in \mathbb{N}$, as discrete-time output at discrete time instant k. The type of these values is $\mathcal{I}$. Component $hold_{\mathcal{O},T}$ takes a discrete-time stream of type $\mathcal{O}^{\mathbb{N}}$ and extends its value at every discrete time instant k over the dense time interval $[k \cdot T, (k+1) \cdot T)$ by holding it constant. Figure 5.7 exemplarily depicts the effect of

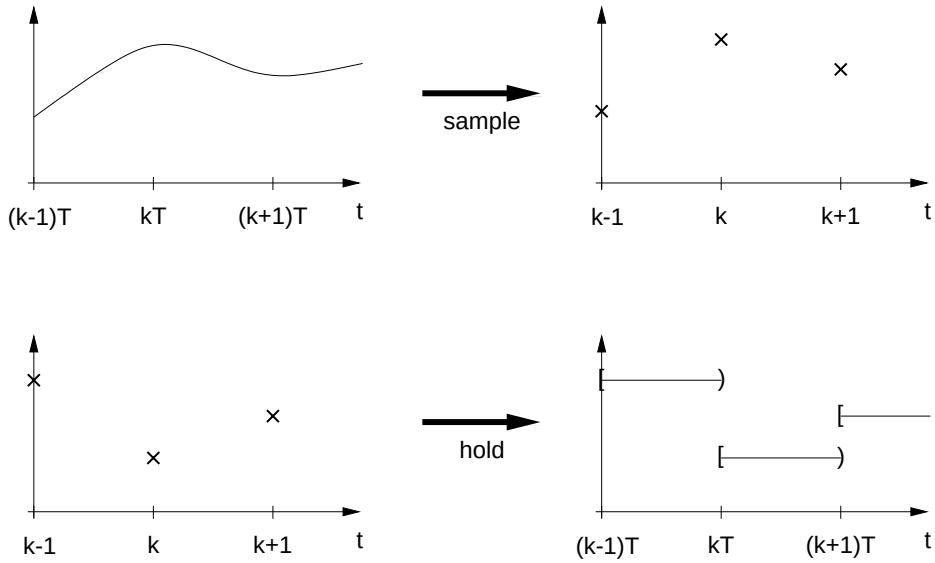
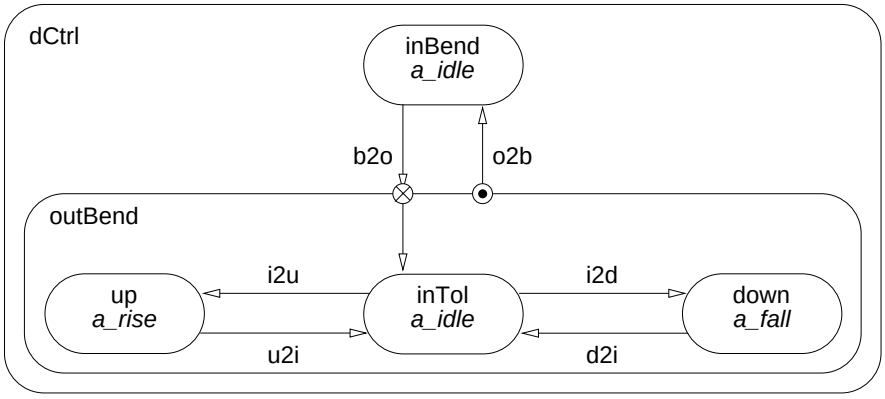


Figure 5.7: Exemplary behavior of *sample* and *hold*.

the *sample* and *hold* components. The components are defined as follows:

$$\begin{array}{lcl} sample_{\mathcal{I},T} & \in & \mathcal{I}^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathcal{I}^{\mathbb{N}}) \\ sample_{\mathcal{I},T}(\iota)(k) & = & \{\iota(k \cdot T)\} \\ hold_{\mathcal{O},T} & \in & \mathcal{O}^{\mathbb{N}} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_{p+}}) \\ hold_{\mathcal{O},T}(o)(t) & = & \{o(\lfloor \frac{t}{T} \rfloor\} \end{array}$$

where $T > 0$ and $\lfloor \frac{t}{T} \rfloor$ denotes the integer part of the division $\frac{t}{T}$. These components define how the logical time axis $\mathbb{N}$ of a DiSChart is mapped to the physical time axis $\mathbb{R}_+$. The mapping depends on constant T which can be regarded as sampling rate. As $sample_{\mathcal{I},T}$ and $hold_{\mathcal{O},T}$ are deterministic, their output upon an input is a singleton set, we therefore also write $\beta = sample_{\mathcal{I},T}(\alpha)$ instead of $\beta \in sample_{\mathcal{I},T}(\alpha)$ and similarly for $hold_{\mathcal{O},T}$.



Actions:

$b2o$	$\equiv$	$bend?$
$o2b$	$\equiv$	$b2o$
$i2u$	$\equiv$	$fHeight.val` \leq lb$
$i2d$	$\equiv$	$fHeight.val` \geq ub$
$u2i$	$\equiv$	$fHeight.val` \geq lb + c \ \wedge \ reset!$
$d2i$	$\equiv$	$fHeight.val` \leq ub - c \ \wedge \ reset!$

Invariants:

$dCtrl_{inv}$	$\equiv$	$True$
$inBend_{inv}$	$\equiv$	$bend.t = bend.t` \ \vee \ now \ mod \ T \neq 0$
$outBend_{inv}$	$\equiv$	$inBend_{inv}$
$inTol_{inv}$	$\equiv$	$fHeight.val` \in (lb, ub) \ \vee \ now \ mod \ T \neq 0$
up_{inv}	$\equiv$	$fHeight.val` < lb + c \ \vee \ now \ mod \ T \neq 0$
$down_{inv}$	$\equiv$	$fHeight.val` > ub - c \ \vee \ now \ mod \ T \neq 0$

(Discrete-time) activities:

a_idle	$\equiv$	$\{ \ aHeight' = aHeight \ \}$
a_rise	$\equiv$	$\{ \ aHeight' \in [aHeight + cp_- \cdot T, aHeight + cp_+ \cdot T] \ \}$
a_fall	$\equiv$	$\{ \ aHeight' \in [aHeight + ev\_T, aHeight + ev_+T] \ \}$

Figure 5.8: DiSChart for the controller.

5.3.6 DiSChart for the EHC's Controller

This section describes the DiSChart for a discrete-time version of the EHC's controller from Section 5.2.5. As we will see in the following section, it is a formal refinement of the relaxed HySChart given there for an appropriate sampling rate T.

Example 5.3 The discrete-time controller is given by the DiSChart in Figure 5.8. The actions are the sames as those in the relaxed HySChart of Section 5.2.5. However, with each action body, $rs' = true$ is implictly associated (see Section 5.3.2). The invariants are also similar to those in the relaxed HySChart. They can be regarded as

resulting from the HySChart's exact invariants by adding disjunction $now \ mod \ T \neq 0$, where T is the constant denoting the sampling rate with which the DiSChart is supposed to be operated. Hence, at sampling times a transition must be taken, if an action guard is true.[10] For the invariant of hierarchic state *dCtrl* observe that $True \ \vee \ now \ mod \ T \neq 0$ is equivalent to $True$.

For the definition of activities, we followed the conventions described above and only explicitly specified the case for $rs \neq init$ and omitted how rs is changed. Each activity is specified by a single computation law here. Provided $rs \neq init$ activity *a_idle* denotes that *aHeight* remains constant. The computation laws for activities a_rise and a_fall contain a nondeterministic assignment. For a_rise the value of *aHeight* is incremented by a value in $[cp_- \cdot T, cp_+ \cdot T]$, where $cp_- < cp_+$ are positive constants and T is as above. Note that this simple kind of activity corresponds to the simple differential constraints specified in the HySCharts of Section 5.2.5. Activity a_fall is similar. Here, *aHeight* is decremented. Making the treatment of rs explicit, a_rise is given by the following computation law:

$$(rs \neq init \ \Rightarrow \ aHeight' \in [aHeight + cp_- \cdot T, aHeight + cp_+ \cdot T] \ \wedge \ rs' = false) \ \wedge$$
$$(rs = init \ \Rightarrow \ aHeight' = aHeight \ \wedge \ rs' = true)$$

Thus, *aHeight* remains unchanged if $rs = init$ and, as the activity is associated with a primitive node, rs is set to *true* in the initialization case. Otherwise it is set to *false*. Unfolding our conventions for the other activities yields a similar result, since they are also associated with primitive nodes.

A stricter version of activity a_rise (an similarly of a_fall) can be obtained by specifying it with the set of computation laws $\{aHeight' = aHeight + cp \cdot T \mid cp \in [cp_-, cp_+]\}$. For the resulting activity, the selected rate with which *aHeight* rises must remain constant as long as no transition is taken. After a transition, the selection of a computation law with a new rate is allowed because of the use of rs and the definition of $da(.)$ for activities (see Section 5.3.3). Depending on the application, this stricter form may be desirable and motivates that a set of computation laws may be used to define an activity in DiSCharts.

In comparison to the state space of the relaxed HySChart of Figure 5.5 the DiSChart's state space is bigger, because it uses the further variable rs. Its state space is $4 \cdot (\mathbb{R} \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times \{init, true, false\})$ corresponding to the four primitive nodes, and to the output variables *aHeight* and *reset*, the latched time stamps for *bend* and *fHeight*, and *now* and rs. Note that the latched time stamp of *fHeight* is not used in the DiSChart, because there is no action guard which senses discrete jumps on the hybrid input channel *fHeight*.

To use the discrete-time component defined by a DiSChart within a (continuous-time) HyAChart, its input interface is preceded by a *sample* component and a *hold* component is added after its output interface. With this extension to dense streams

[10] Remember that the exact invariant results from the negated action guards.

the component defined by the above DiSChart can be used to replace the continuous-time component *Control* of the HyAChart for the EHC given in Figure 3.2, left. □

5.4 Time Refinement of HySCharts

This section first identifies sufficient conditions under which a DiSChart is a refinement of a relaxed HySChart (Section 5.4.1). Then, we explain how these conditions can be satisfied. This amounts to introducing methods which lead from the relaxed discrete part *RCom* and the relaxed analog part *RAna* of a relaxed HySChart to discrete-time versions of them such that the resulting DiSChart is a refinement of the relaxed HySChart. For the analog part, the application of methods from numerical mathematics and control theory is discussed. Finally, the proposed methods are applied to an example and a further field of application is indicated. Note that the resulting DiSChart may, but need not, be deterministic. If it is nondeterministic, it can serve as basis for further development steps with underlying discrete time model. Otherwise, it can directly be executed.

5.4.1 Refinement Conditions

Preliminaries. Usually some knowledge about a component's input is necessary in order to find a DiSChart that refines a given HySChart, which specifies the component. For instance, Lipschitz constants constraining the evolution on continuous input channels are needed to find a sampling rate for a DiSChart such that the DiSChart detects boundary crossings on these channels with the precision prescribed in the HySChart. Furthermore, when constructing a discrete-time implementation of a relaxed HySChart we want to regard the discrete and the analog part largely in isolation, because this enables us to employ standard techniques for the analog part. This motivates to restrict the set of considered inputs and the set of considered evolutions of the state. In the following theorem we will use $I \subseteq \mathcal{I}^{\mathbb{R}_{p+}}$ for the set of considered inputs and $S \subseteq (n \cdot \mathcal{S}_C)^{\mathbb{R}_{p+}}$ for the set of considered evolutions of the state, where $n \cdot \mathcal{S}_C$ is the state space of the relaxed HySChart. Typically, I and S may be described by evolution constraints like Lipschitz constants or minimum event separations (Section 3.4). The method which is introduced in Section 5.4.2 to derive a sampling implementation for a HySChart's discrete part is based on the following evolution constraints: For each discrete and hybrid input channel the minimum event separation must be given, and for each hybrid and continuous input channel and for each controlled variable a Lipschitz constant l and an associated error e, if any, must be given. The Lipschitz constant l is interpreted as constraining every period of ideal continuous evolution. The actual evolutions, which are assumed, result from allowing the error e at every point in time, i.e. the ideal continuous evolutions are superimposed by the error. The following example illustrates these constraints.

Example 5.4 (Evolution constraints.) As an example consider the real-valued hybrid input channel x. By convention such a channel is encoded as a stream of values and time stamps such that at any time instant the time stamp denotes the last time when there was a discontinuity on the channel (Section 3.4). The convention for discrete channels and channels only transmitting events is similar. To simplify notation, we denote the set of streams which satisfy these conventions for a channel of type D by $cs(D)$. Formally, $cs(D)$ is defined as follows:

$$\begin{aligned} cs(D) \;=\; \{\sigma \in (D \times \mathbb{R}_+)^{\mathbb{R}_{p+}} \mid \forall u, v \in \mathbb{R}_+ . \, \sigma.t(v) = u \Rightarrow \\ u \leq v \;\wedge\; \sigma|_{[u,v]} \in (D \times \mathbb{R}_+)^{\mathbb{R}_{s+}}|_{[u,v]}\} \end{aligned}$$

where $(D \times \mathbb{R}_+)^{\mathbb{R}_{s+}}$ are the smooth functions from $\mathbb{R}_+$ to $D \times \mathbb{R}_+$, $\sigma.t$ denotes the projection of σ on the stream of time stamps and $\sigma.val$ denotes the projection on the stream of values. Event channels do not transmit values, but only time stamps. For the set of streams of time stamps satisfying the convention for event channels we write $cs(E)$.[11]

For channel x, we assume that the minimum event separation is m, the Lipschitz constant constraining continuous periods of evolution is l and error e is associated with the channel. Then the set $I \subseteq (\mathbb{R} \times \mathbb{R}_+)^{\mathbb{R}_{p+}}$ of considered evolutions for x is defined by:

$$\begin{aligned} I \;=\; \{\tau \in cs(\mathbb{R}) \mid \exists \sigma \in cs(\mathbb{R}). \, \sigma.t = \tau.t \;\wedge\; mes(\sigma) \leq m \;\wedge \\ Li(\sigma) \leq l \;\wedge\; \forall u \in \mathbb{R}_+ . \, |\sigma.val(u) - \tau.val(u)| \leq e\} \end{aligned}$$

Functions $mes(.)$ and $Li(.)$ are defined in Section 3.4. □

For controlled variables the minimum event separation is not needed, because discrete changes in them are initiated by the regarded component's discrete part, not by its environment. Thus, the minimum event separation is a commitment rather than an assumption in this case. We remark that the way errors and Lipschitz constants are interpreted suits well to the effects introduced by the output relaxation R_{int} in relaxed HySCharts. Therefore, input complying with such constraints can, e.g., be provided by relaxed HySCharts.

Refinement theorem. Informally, a DiSChart is a refinement of a HySChart, if the following holds: First, the next states produced by the DiSChart's discrete part must also be valid next states of the relaxed HySChart's discrete part. Second, over the time periods where the DiSChart's discrete part idles the HySChart's discrete part must also be able to idle. Third, during these idle periods the output of the DiSChart's analog part must be a possible output of the relaxed HySChart's analog part at sampling instants, and between sampling instants, the *hold* extension of the DiSChart's output must be in the output relaxation of the HySChart's output.

In the following theorem we formalize this idea. Let $DCmp \in n \cdot \mathcal{S}_D \to (\mathcal{I}^{\mathbb{N}} \to \mathcal{P}(\mathcal{O}^{\mathbb{N}}))$ be the semantics of a discrete-time component specified with a DiSChart. We write

[11] Note that E is introduced as the neutral element for the Cartesian product in Section 3.3.2.

DCom and *DAna* to refer to its discrete and analog part, respectively, and $DOut^\dagger$ for its time extended output projection. Let $Cmp \in n \cdot \mathcal{S}_C \to \mathcal{I}^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_{p+}})$ be the semantics of a continuous-time component specified with a relaxed HySChart. We write *RCom* and *RAna* for its discrete and analog part, $ROut^\dagger$ for its time extended output projection and R_{int} for its output relaxation. Refinement w.r.t. all inputs in I is denoted by $\preceq_I$, i.e. $A \preceq_I B$ iff $\forall \iota \in I.\, A(\iota) \subseteq B(\iota)$ for continuous-time components A and B. For state (or state stream) s we write $s.x$ in the following to denote the value (or the evolution) of variable x in that state (or stream). The Cartesian projection of a space X on space Y is denoted by π_Y, where $X = Y \times Z$ (up to isomorphism) for some space Z. Space Y is also called a *factor space* of space X. Projection is overloaded to arbitrary spaces of which Y is a factor space. Therefore, X does not occur as index in π_Y. For program-state spaces $n \cdot \mathcal{S}_a$ and $n \cdot \mathcal{S}_b$, projection is defined by $\pi_{n\cdot\mathcal{S}_b}((k,s)) = (k, \pi_{\mathcal{S}_b}(s))$, where $(k,s) \in n \cdot \mathcal{S}_a$ and $\mathcal{S}_b$ is a factor space of $\mathcal{S}_a$. Its type is $(n \cdot \mathcal{S}_a) \to (n \cdot \mathcal{S}_b)$. Projection is extended to streams and sets of streams in a pointwise manner. The data-state space $\mathcal{S}_C$ of *Cmp* is required to be a factor space of that part of the data-state space $\mathcal{S}_D$ of *DCmp* which is not needed by the DiSChart for variable *rs* and for further auxiliary variables used in its numerical computations (see Section 5.3.3). This ensures that the part of the data-state space which both components have in common is not affected by initialization of *DAna*.

Theorem 5.1 (Discrete-time refinement of HySCharts.) *For set I of considered inputs, $I \subseteq \mathcal{I}^{\mathbb{R}_{p+}}$, start states s, s' with $s.rs = init$, $s.now = 0$ and sampling period T the time extended $DCmp(s)$ refines $Cmp(s')$, formally*

$$sample_{\mathcal{I},T};_\times\ DCmp(s);_\times\ hold_{\mathcal{O},T} \preceq_I Cmp(s')$$

holds, if there is a set $S \subseteq (n \cdot \mathcal{S}_C)^{\mathbb{R}_{p+}}$ of evolution constraints for the state stream such that the following holds:[12]

1. *$\mathcal{S}_C$ must be a factor space of the part of $\mathcal{S}_D$ which is not needed for auxiliary variables of DAna and for variable rs, furthermore $s' = \pi_{n\cdot\mathcal{S}_C}(s)$*
2. *at sampling instants DCom refines RCom:*

   $$\forall i, u, v.\ \ u.now \bmod T = 0 \ \wedge\ v \in DCom(i,u) \Rightarrow \pi_{n\cdot\mathcal{S}_C}(v) \in RCom(i, \pi_{n\cdot\mathcal{S}_C}(u))$$

3. *provided the input and the state evolve according to the assumed restrictions $I \subseteq \mathcal{I}^{\mathbb{R}_{p+}}$ and $S \subseteq (n \cdot \mathcal{S}_C)^{\mathbb{R}_{p+}}$, and the value of the local clock, now, is correct, RCom can remain idle during a sampling interval if DCom was idle at the beginning of the interval:*

   $$\begin{aligned}
   &\forall k \in \mathbb{N}.\, \forall \iota \in I|_{[kT,(k+1)T)}.\, \forall \sigma \in S|_{[kT,(k+1)T)}.\, \forall s \in n \cdot \mathcal{S}_D.\\
   &\quad \pi_{n\cdot\mathcal{S}_C}(s) = \sigma(kT) \ \wedge\\
   &\quad \forall t \in \mathbb{R}_+.\, \sigma.now(t) = t \ \wedge && \textit{(now is correct)}\\
   &\quad s \in DCom(\iota(kT), s) \Rightarrow && \textit{(DCom idles)}\\
   &\quad \forall t \in [k \cdot T, (k+1) \cdot T).\, \sigma(t) \in RCom(\iota(t), \sigma(t)) && \textit{(RCom idles)}
   \end{aligned}$$

[12] Note that this refinement relation is a *U-simulation* in the sense of [Bro97b].

4. *On any interval where DAna is only reset at the beginning, DAna refines the trajectories in RAna at sampling instants and the extension of its output trajectories by hold refines the output trajectories in RAna relaxed by* R_{int}.
Formally, for every interval $\delta = \{k_1 + 1, \ldots, k_2 + 1\}$, $k_1 \leq k_2$ *in* $\mathbb{N}$, *the following must hold:*

$$
\begin{array}{ll}
\forall \iota \in I.\, \forall \iota' \in \mathcal{I}^{\mathbb{N}}.\, \forall \tau' \in (n \cdot \mathcal{S}_D)^{\{k_1,\ldots,k_2+1\}}. & \\
\quad \iota' \in sample_{\mathcal{I},T}(\iota) \ \wedge & \\
\quad \tau'.rs(k_1) = true \ \wedge & \textit{(reset at beginning)} \\
\quad \forall k \in \delta.\, \tau'.rs(k) = false \ \wedge & \\
\quad (\iota'|_\delta, \tau'^1|_\delta, \tau'|_\delta) \in DAna|_\delta & \textit{(τ' satisfies DAna)} \\
\quad \Rightarrow & \\
\quad \exists \tau \in (n \cdot \mathcal{S}_C)^{[k_1 T,(k_2+1)T)} \cap C. & \\
\qquad (\iota|_{[k_1 T,(k_2+1)T)}, \tau, \tau) \in RAna|_{[k_1 T,(k_2+1)T)} \ \wedge & \textit{(τ satisfies RAna)} \\
\qquad \forall k \in \{k_1, \ldots, k_2\}.\, \pi_{n\cdot\mathcal{S}_C}(\tau'(k)) = \tau(kT) \wedge & \textit{(sampling instants OK)} \\
\qquad \pi_{n\cdot\mathcal{S}_C}(\tau'(k_2+1)) = \lim_{x \nearrow (k_2+1)T} \tau(x) \ \wedge & \textit{(last value OK)} \\
\qquad hold_{\mathcal{O},T}(DOut^\dagger(\tau')) \subseteq R_{int}(ROut^\dagger(\tau)) & \textit{(hold extension OK)}
\end{array}
$$

where τ'^1 *is the right shift of* τ' *by one time unit and* C *is the set of continuous functions in* $(n \cdot S_C)^{[k_1 \cdot T,(k_2+1)\cdot T)}$.

5. *set* S *of constraints on the state evolution indeed constrains the trajectories produced by RAna for inputs in* I: $(\iota, \tau, \tau) \in RAna|_\delta \Rightarrow \tau \in S|_\delta$ *for all* $\iota \in I|_\delta$ *and all intervals* δ

If an analog part *RAna* and its discrete-time implementation *DAna* satisfy the conditions of assumption 4 of the theorem, we also say that *DAna* is a *sample-and-hold-refinement* of *RAna*, and likewise for activities.

The proof of the theorem proceeds inductively and is based on the operational hybrid step relations defined for HySCharts and DiSCharts in Appendix A.1.2 and A.2.1, respectively. Further main properties needed in the proof are

- that the relaxed analog part always determines the correct value of *now*, if it was initialized correctly (Section 5.2.2),
- that on intervals where *rs* in DiSCharts is *true* initially and *false* thereafter, the control state does not change (Section 5.3.3),
- that the initialization step of *DAna* does not affect the state space of the HySChart and sets *rs* to *true* (assumption 1 together with Section 5.3.3).

The proof is given in Appendix A.2.2.

Correct values of the local clock (in assumption 3) are needed, because the triggering of transitions depends on them. Assumption 4 only needs to regard intervals where the discrete part idles, because the case when it is not idle is covered by assumption 2. Idling during a time period in the discrete time case means that *rs* is (at most) *true* initially and *false* thereafter. Furthermore, the control state is constant during

an idle period. If it was not, this would mean that *DCom* takes a transition which implies that it sets *rs* to *true*.

Remarks. Assumption 4 in the theorem relating *DAna* and *RAna* may be difficult to establish in cases where jumps in the external input ι directly affect the analog part, because in this case *DAna* makes its next step without knowing about the jump whereas *RAna* was able to change its activity at the time instant the jump occurred. In general this reflects that systems should be designed such that jumps in the external input do not directly affect the analog part. However, we can weaken assumption 4 of the theorem under certain circumstances: If the discrete part is guaranteed to react in a way that resets those variables affected by the jump in the input such that their new value determined by the discrete part is the same in the DiSChart and the HySChart, we can permit that the next state produced by *DAna* is not a valid output of *RAna*. We do not formalize this further.

Apart from this difficulty, finding a τ which satisfies *RAna* and which corresponds to τ' as required in assumption 4 is not a major problem, if *RAna* is liberal enough and *DAna* is chosen sensibly. Methods to find *DAna* are discussed in Section 5.4.3.

The theorem could also be stated for other initial states s with $s.now = kT$ (for a $k \in \mathbb{N}$) and consistent values of those private variables which are only needed by *DAna*. We refrain from doing so, because this would complicate the proof of the theorem while not providing further insight.

Theorem 5.1 provides a joint refinement of *RCom* and *RAna*. While fully isolated refinement of *RCom* and *RAna* would also be possible, we focused on this variant, because it does not require modifications in the machine model which would otherwise be necessary in order to decouple *RCom* and *RAna*. In Section 5.6 we outline how isolated time refinement of the discrete part can be realized.

5.4.2 Sampling for the Discrete Part

Here we explain how the discrete part of a DiSChart which satisfies assumptions 2 and 3 of the refinement theorem from above can be constructed from a relaxed HySChart. For the construction, the relaxed invariants as well as the exact invariants from which they were derived are needed.[13] The HySChart with the exact invariants must be total, i.e. it cannot refuse an input. Furthermore, evolution constraints for all those input channels and controlled variables must be given whose values occur in the exact invariants. The construction can be performed automatically, if all this data is given and if the arithmetic expressions in the exact invariants are sufficiently simple.

[13] Remember that the exact invariant of a node is defined as the negation of the disjunction of the actions guards of the transitions directly emerging from the node (and from none of its subnodes) (Section 3.5.1).

Figure 5.9 depicts the underlying intention for this construction and also for the construction of the (discrete-time) analog part in the next section. For a given exact HySChart and its relaxation, a DiSChart is sought which refines the relaxed HySChart (solid line from the relaxed HySChart to the DiSChart in the figure). In the construction, elements from the exact HySChart are used in order to ensure that the behavior of the DiSChart closely resembles it (indicated by the dashed line in the figure). These used elements are the exact invariants and the exact activities. The structure of the hierarchic graphs defining the exact HySChart, the relaxed HySChart and the DiSChart is the same. Syntactically, the three diagrams only differ in their actions, invariants and activities.

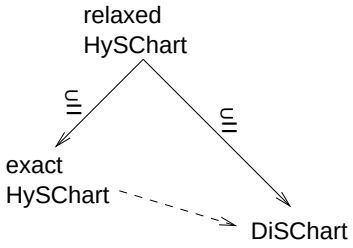


Figure 5.9: Intended relation between the exact HySChart, the relaxed HySChart and the DiSChart.

Idea. Methods for the construction of the analog part are considered in the next section. The principal idea in the construction of the discrete part is as follows. If we want that the discrete part running with a certain clock rate refines the relaxed discrete part which is not bound to a fixed time grid, we must ensure that the clock rate is chosen such that a sampling step falls in every interval during which the relaxed discrete part has the choice to idle or to take a transition and after which a transition must be taken. A sufficient condition is to ensure that a sampling step lies in every interval in which a (subformula of a) relaxed invariant and the corresponding action guard, from which it was derived, are simultaneously true. In other words, we must choose a sampling rate T such that the smallest tolerance permitted by the relaxation of the HySChart can be satisfied. When a transition guard is true at a sampling instant, it must be executed, because idling and waiting until the next sampling instant may cause that guards and invariants in the relaxed discrete part are both violated for the state at that time.

In the following we write $DCom$ for the DiSChart's discrete part and $RCom$ for the relaxed HySChart's discrete part. For the construction of $DCom$ we start with the same hierarchic graph which defines the HySChart. Only the invariants and the action bodies in the graph will be modified. Thus, the DiSChart and the HySChart have the same set of control states. The data-state space of the DiSChart contains the further controlled variable rs and possibly some further private variables which are only needed by the computation laws defining the DiSChart's analog part. As before,

we write $n \cdot \mathcal{S}_D$ (or $n \cdot \mathcal{S}_C$) for the DiSChart's (or HySChart's) program state space. $\mathcal{S}_C$ is a factor space of $\mathcal{S}_D$.

Evolution Constraints. In order to derive a sampling implementation for the discrete part we need some assumptions on the dynamics of those input channels and controlled variables which occur in the exact invariants:

- For every discrete or hybrid input channel, discrete jumps in them must be correctly encoded in their associated time stamps, and the minimum event separation for each such channels must be given.
- For every hybrid input channel, a Lipschitz constant l and an associated error e must be given. In any time interval during which the time stamp information on a hybrid channel does not change, the evolution of values on the channel may be not more than e away from an evolution which is Lipschitz continuous with Lipschitz constant l.
- For every continuous input channel and every controlled variable a Lipschitz constant l and an associated error e must be given. The streams on a continuous channel must always evolve according to these constants. The controlled variables must evolve according to these constants during any time interval in which they are continuous.

Example 5.4 shows how these constants formally result in sets $I \subseteq \mathcal{I}^{\mathbb{R}_{p+}}$ and $S \subseteq (n \cdot \mathcal{S}_C)^{\mathbb{R}_{p+}}$ of evolution constraints for the input and state. We will need these sets when we show that the constructed discrete part indeed satisfies assumptions 2 and 3 of Theorem 5.1.

The method developed in the following paragraph results in a set of inequations depending on the sampling rate, the evolution constraints and the constants used in the relaxation of the HySChart. Thus, if two of these classes of quantities are given, we can compute constraints on the remaining one from the inequations. An application of this kind is discussed in Section 5.4.5.

Deriving the sampling rate. We derive a set of constraints on the sampling rate T, with which $DCom$ must be operated, from the relaxed invariants of every (primitive or hierarchic) node of the HySChart. The relaxed invariants are assumed to result from the relaxation of exact invariants defined in Section 5.2.1. Hence, each invariant is a finite conjunction of finite disjunctions of atoms, where each atom is a comparison of arithmetic expressions, a comparison of time stamps, a comparison of a time stamp and variable now, or another proposition over variables with domains different from the real numbers (see the bottom three lines in the definition of the relaxation in Section 5.2.1). The atoms, which are of interest here, are comparisons of variable now with time stamps and inequalities involving arithmetic expressions over controlled variables and the input on hybrid and continuous channels. For every atom of the form $now - x.t^{\backprime} < \varepsilon_x$ for a variable x associated with discrete or hybrid input channel, we add constraint $T \leq \varepsilon_x$. This ensures that there is a sampling point between a

discrete change in x and the permitted delay ε_x after the change.[14] For every atom $a(h.\vec{val}\backslash, c.\vec{val}\backslash, \vec{v}) < \varepsilon_a$, we add constraint $T \leq \frac{\varepsilon_a - 2 \cdot \varepsilon_{err,a}}{l_a}$, where a is an arithmetic expression over tuples of current input variables $h.\vec{val}\backslash, c.\vec{val}\backslash$ associated with hybrid and continuous input channels, respectively, and over a tuple of controlled variables $\vec{v}$. Constant l_a is the Lipschitz constant constraining the ideal evolution of expression a, ε_a limits the permitted overshooting of a over threshold 0 and $\varepsilon_{err,a}$ is an error that is associated with the computation of a and which affects its ideal evolution. For sufficiently simple arithmetic expressions, like addition, l_a can be derived from the Lipschitz constants constraining the ideal evolution of the variables occurring in the expression, see e.g. [Kön90]. These constants are required to be given. Constant $\varepsilon_{err,a}$ results from the errors that are associated with the ideal values of all variables occurring in the expression. E.g. for controlled variables, their discretization error has to be considered here. Error estimation techniques from numerical mathematics allow us to derive the possible error in a from the errors associated with the variables occurring in it [HH91]. Bounds for these errors are assumed given. For a sampling implementation $\varepsilon_a - 2 \cdot \varepsilon_{err,a}$ must be greater than zero, because T must be positive. The condition $T \leq \frac{\varepsilon_a - 2 \cdot \varepsilon_{err,a}}{l_a}$ provides that whenever the value of expression a evolves continuously, there is a sampling point in every time interval during which a can cross the interval $[0, \varepsilon_a)$. In this interval, the relaxed invariant still is true although the exact invariant $a < 0$ already is false. To show this we derive a lower bound on the time it takes for the computed value of a to rise from 0 to ε_a. Assume the computed value of a is 0 at time t_1 and ε_a at time t_2, $t_2 > t_1$. Furthermore, assume the worst case for the error. In this case, the exact value of a, denoted by $\overline{a}$, lies $\varepsilon_{err,a}$ above the computed one at time t_1 and $\varepsilon_{err,a}$ below the computed one at time t_2. Then, $\varepsilon_a = a(t_2) - a(t_1) = \overline{a}(t_2) + \varepsilon_{err,a} - (\overline{a}(t_1) - \varepsilon_{err,a}) \leq l_a |t_2 - t_1| + 2 \cdot \varepsilon_{err,a}$ holds, where we used Lipschitz continuity. Thus, the value of a needs at least time $\frac{\varepsilon_a - 2 \cdot \varepsilon_{err,a}}{l_a}$ to rise continuously from 0 to ε_a. Choosing T less or equal to this constant ensures that we can detect whenever a crosses the interval $[0, \varepsilon_a)$.

In the rest of this section we assume that $T > 0$ is chosen such that it satisfies all the constraints on it. If $\varepsilon_a - 2\varepsilon_{err,a} > 0$ for every arithmetic expression a, this is possible since each constraint limits T from above with some value greater 0 and there are only finitely many constraints. (The number of constraints is finite, because there are only finitely many atoms in finitely many action guards in a HySChart, leading to finitely many atoms in finitely many (relaxed) invariants.)

Actions and invariants in *DCom*. To obtain a refinement of the relaxed discrete part, we start with replacing every relaxed invariant in the HySChart with the exact invariant from which it was derived. Then, we weaken the invariant of every (primitive or hierarchic) node in the resulting HySChart by adding disjunct $now \bmod T \neq 0$ to it. This causes that the new invariant can only remain true at sampling points

[14] Remember that $\varepsilon_x > 0$ should be chosen smaller than the minimal separation between two discrete changes for x to avoid that discrete changes are not detected.

kT, $k \in \mathbb{N}$, if the exact invariant is true. This, in turn, requires that no outgoing transition of the considered node is enabled, because exact invariants result from negating action guards. Thus, the exact invariants with the additional disjunct express that a transition must be taken at the first sampling point where it is enabled.

Finally, we add conjunct $rs = true$ to the action body of every transition in the chart. The graph with the modified invariants and transitions defines our DiSChart. $DCom$ is directly derived from it in the way explained in Section 3.5.1 for HySCharts.

$DCom$ is total if the discrete part of the HySChart with exact invariants, denoted by Com, was, which we required at the beginning of this section. This follows by case distinction. For any current input i and latched program state s, if $s.now \ mod \ T = 0$ $DCom$ and Com coincide, $DCom(i, s) = Com(i, s)$. If we are not at a sampling point, i.e. $s.now \ mod \ T \neq 0$, all invariants are true for s and $DCom$ therefore can idle, $s \in DCom(i, s)$. Note that it is sensible to require that the HySChart with exact invariants it total, because we regard it as describing the "ideal" behavior, while the HySChart with relaxed invariants defines the allowed approximating behavior. As the relaxed HySChart is an abstraction of the one with exact invariants, it is also total if the exact one is.

Proof of correctness. We now prove that $DCom$ as constructed above satisfies assumptions 2 and 3 of the Theorem 5.1. Assumption 2 is simple: The extended action bodies do not affect the state space $n \cdot \mathcal{S}_C$ of the HySChart. Furthermore, at sampling instants the additional disjunct added to the exact invariants is false. Hence, at sampling instants $DCom$ projected on state space $n \cdot \mathcal{S}_C$ behaves like Com, and Com itself refines $RCom$, since $RCom$ is derived from Com by weakening invariants. Weakening an exact invariant means that the set denoting the exact invariant's semantics is a subset (and hence a refinement) of the semantics of the relaxed invariant. As additive hierarchic graphs are compositional w.r.t. refinement (Section 3.3.5), this yields that the graph with the exact invariants refines the graph with the relaxed invariants.

For assumption 3 we have to prove that $RCom$ can idle during a sampling interval, if $DCom$ was idle at the beginning of the interval and the input and the state evolve according to the constraints $I \subseteq \mathcal{I}^{\mathbb{R}_{p+}}$ and $S \subseteq (n \cdot \mathcal{S}_C)^{\mathbb{R}_{p+}}$ assumed for them. In other words, no transition in $RCom$ is enforced between two sampling instants of $DCom$. Let $k \in \mathbb{N}, \iota \in I|_{[k \cdot T,(k+1) \cdot T)}$ and $\sigma \in S|_{[k \cdot T,(k+1) \cdot T)}$ with correct time in σ, formally $\forall t \in \mathbb{R}_+ . \sigma.now(t) = t$. Furthermore, let $s \in n \cdot \mathcal{S}_D$ with $\pi_{n \cdot \mathcal{S}_C}(s) = \sigma(kT)$ such that $DCom$ can idle for $\iota(kT)$ and s. By construction of the discrete part from the hierarchic graph for a DiSChart or HySChart, respectively, (Figures 3.15 and 3.16) idling means that for the current input $\iota(kT)$ and latched state s the invariant of the primitive node which is encoded in the control-state information in s and the invariants of all its supernodes are true.[15]

[15] The hierarchic nodes which directly or via further subnodes contain a primitive node are called its *supernodes*.

Apart from the condition $now \ mod \ T \neq 0$ which is false for s, the invariants in $DCom$ coincide with those in the exact discrete part. Hence, omitting this condition on now, they are a finite conjunction of finite disjunctions over (negated) comparisons of arithmetic expressions, (negated) comparisons of time stamps and further (negated) propositions. Schematically each invariant can be written as follows: $\bigwedge_{i=1}^{g}(\bigvee_{j=1}^{h_i} at_{i,j})$ where each $at_{i,j}$ either is a (negated) comparison of arithmetic expressions, a (negated) comparison of time stamps, or another (negated) proposition over variables with domains different from the real numbers. The structure of the corresponding relaxed invariant is: $\bigwedge_{i=1}^{g}(\bigvee_{j=1}^{h_i} r_{i,j})$ where depending on $at_{i,j}$, $r_{i,j}$ is one of the following:

- $a(h.\vec{val}\backslash, c.\vec{val}\backslash, \vec{v}) < \varepsilon_a \ \vee \ \bigvee_{\ell=1}^{m} now - h_\ell.t\backslash < \varepsilon_{h_\ell}$ if $at_{i,j}$ is a comparison of arithmetic expressions not involving time stamps and now,
- $ts\backslash = ts \ \vee \ now - ts\backslash < \varepsilon_{ts}$ if $at_{i,j}$ is a comparison of (latched and current) time stamps for an input channel,
- $\neg p(d.\vec{val}\backslash, \vec{v}) \ \vee \ \bigvee_{\ell=1}^{m} now - d_\ell.t\backslash < \varepsilon_{d_\ell}$ if $at_{i,j}$ is a proposition p over variables that are not real-valued.

Let z denote the primitive node which corresponds to the control-state information in s. As $DCom$ can idle for $\iota(kT)$ and s, the invariant of node z and the invariants of all its supernodes must be true. Since the invariants are given as a conjunction of disjunctions, this means that one of the atoms in each disjunction must be true. We proceed by case distinction over such atoms at which are true for s and $\iota(kT)$, and show that their relaxation r remains true until the next sampling instant for ι and σ.

First, assume at is a (negated) comparison of time stamps or a (negated) proposition p over variables that are not real-valued. If there is no jump on any of the input channels whose value is used in at, then all the variables associated with these input channels and also all the controlled variables occurring in at evolve continuously. As the discrete topology is used for all these variables, this means that they remain constant. Hence, at is guaranteed to remain true and its relaxation r therefore also remains true. Now we assume that at time $u > kT$ there is a jump on input channel c whose value is used in at. In this case, the jump is signaled by time stamp u as the current time stamp for that channel.[16] As $\sigma.now(u)$ also is u and variable now evolves in pace with physical time, the difference of now and u will be less than T at the end of the sampling interval. Due to the relaxation, there is an atom $now - c.t < \varepsilon_c$ in r. By the choice of T it is less or equal to ε_c. Hence, this atom will remain true until the end of the sampling interval and make r true, as r is a disjunction. Second, assume at is a (negated) comparison of arithmetic expressions, $at \equiv \neg(a(h.\vec{val}\backslash, c.\vec{val}\backslash, \vec{v}) \geq 0)$. Moreover, assume that there is no jump on any of the input channels whose value is used in at, i.e. they all evolve according to the Lipschitz constants and associated errors which define I and S. Then, by definition of the relaxation, the relaxation r of at remains true as long as the value of arithmetic expression a is less than a given permitted deviation ε_a. Selecting T in the way described above ensures that if the

[16] Note that we require correct time stamps in inputs in I.

value of a is less than 0 for s and $\iota(kT)$ it can not reach the bound ε_a before time $(k+1)T$ (see above). Hence, *RCom* can remain idle. If there is a jump on one of the relevant input channels, the situation is like in the first case for jumps on channels which are relevant in propositions.

This finishes the proof. For all invariants associated with the current control state, we have established that for every atom which is true in such an invariant of *DCom* for $\iota(kT)$ and s, there is a corresponding predicate in the respective relaxed invariant which remains true for input stream ι and state stream σ. This provides that the respective relaxed invariant remains true until the end of the sampling interval. Hence, *RCom* can idle throughout the interval.

Finally, we point out that the minimum event separation for discrete and hybrid input channels is not needed to establish that assumptions 2 and 3 of Theorem 5.1 hold for *DCom* and *RCom*, but rather to ensure the utility of these discrete parts. Namely, discrete jumps on discrete and hybrid input channels may fail to be detected if the minimum event separations for them are not greater than the permitted delay after which a (relaxed) HySChart or a DiSChart has to react to them.

5.4.3 Sampling for the Analog Part

As is the preceding section, we assume an exact HySChart and a relaxed HySChart given in this section. The aim is to develop methods for the construction of the analog part *DAna* of a DiSChart which refines the relaxed HySChart. The DiSChart is supposed to result from the relaxed HySChart by changing the time model, activities, actions and invariants, but leaving the chart's structure unchanged (see also Figure 5.9). The construction of *DAna* will usually also involve information about the exact analog part. This is similar to the construction of *DCom* in which the exact invariants are used (Section 5.4.2).

For the construction of *DAna* methods from numerical mathematics and control theory can be applied. Numerical integration algorithms can be used to compute an approximation of the solution of an initial value problem at a given point in time (or a sequence of points in time) [Sch88]. Control theory offers techniques to obtain discrete-time algorithms which result in behavior that is similar to given continuous-time behavior w.r.t. specific characteristics (see below). Furthermore, direct techniques exist for the design of discrete-time controllers of analog plants [Oga87]. In terms of HySCharts, all these techniques work on the level of individual activities, not on the analog part as a whole. Usually, they build upon a problem specification in terms of an initial value problem or a Laplace transform. In contrast, the analog part in a HySChart consists of the disjoint sum of sequentially composed, adapted activities. Depending on their description the individual activities may be accessible to numerical and control theory techniques. In this section we therefore first clarify under which circumstances the discrete-time refinement of individual activities yields a discrete-time refinement

DAna of the analog part. Then, numerical and control theory techniques are outlined which can be used to construct discrete-time refinements of activities.

Note that the refinement of the analog part leads to further constraints on the sampling rate. In contrast to the construction of *DCom*, automation in the construction of *DAna* is only possible in very specific cases, like in the example of the EHC where the analog dynamis are linear (Section 5.4.4) or in case of specific kinds of activities (Section 5.4.3.3). Usually a desinger will at least have to determine the refinements for the individual activities manually. Nevertheless, this section explains how numerical methods and methods from control theory can support this task.

5.4.3.1 Compositional Discretization

The central result of this section is that a discrete-time refinement of a HySChart's analog part can be obtained by finding discrete-time refinements of all primitive activities in the analog part, provided activities at different levels of hierarchy are completely independent.[17] This is formalized in the following.

Let *RAna* be the relaxed analog part of a HySChart. By its construction from the chart as described in Section 3.5.2 its form is $RAna = +_{i=1}^{n} (;_{+j=1}^{m_i} da(RAct_{i,j}))$, where each $RAct_{i,j}$ is a relaxed activity associated with a primitive or hierarchic node in the chart. Constants n and m_i for $i \in \{1, \ldots, n\}$ reflect the hierarchical structure of the HySChart. Constant n is the number of primitive nodes in the chart and m_i is the *hierarchic level* of primitive node i, i.e. it is the number of supernodes in which the primitive node is contained. For the refinement, we only consider DiSCharts which are structurally equivalent to the HySChart and only differ from it in their time model, activities, actions and invariants. Let *DAna* be the (discrete-time) analog part of such a DiSChart with the same structure as *RAna*, i.e. $DAna = +_{i=1}^{n} (;_{+j=1}^{m_i} da(DAct_{i,j}))$ where the $DAct_{i,j}$ are discrete-time activities. Furthermore, let $n \cdot \mathcal{S}_D$ be the state space of the DiSChart and $n \cdot \mathcal{S}_C$ that of the HySChart. $\mathcal{S}_C$ is required to be a factor space of $\mathcal{S}_D$ and the input and output spaces of the DiSChart and the HySChart must coincide.

Besides this structural compatibility we need that hierarchic activities are completely independent from each other. In detail, we require that in every sequential composition of the activities in *RAna* and *DAna* the set of controlled variables occurring in each of the activity specifications (except for variable *rs*) is disjoint from the set of controlled variables in the other activities of the sequential composition. In *DAna*, variable *rs* may be referenced by every activity. Here, the conventions for the modification of *rs* prevent undesirable interference (cf. Section 5.3.3). For *DAna*, we furthermore require

[17] This is not an immediate consequence of the compositionality of time-extended additive hierarchic graphs w.r.t. refinement. Due to the feedback in the machine model we have to regard the analog part and the activities under feedback. The property we need besides compositionality is a kind of distributivity of feedback over disjoint sum and sequential composition.

that every discrete-time activity $DAct_{i,j}$ references all controlled variables occurring in $RAct_{i,j}$. It may reference additional controlled variables of the DiSChart which are not in the state space of the HySChart, but no further variables of the HySChart. Due to the definition of activities, due to their relaxation and due to the associated conventions (Sections 3.5.2, 5.2.2 and 5.3.3), this ensures that $DAct_{i,j}$ and $RAct_{i,j}$ are the identity on variables not occurring in their definition. Practical modeling examples, like the EHC or [PPS00], show that the independence condition is not always, but often satisfied. If it is not satisfied, it may be possible to modify a model by grouping hierarchical activities together.

Before we proceed, some further notation needs to be introduced. We write $\mathcal{A}_{i,j}$ and $\mathcal{D}_{i,j}$ for the factor space of $\mathcal{S}_C$ and $\mathcal{S}_D$, respectively, built from the product of the domains of those controlled variables occurring in the definition of $RAct_{i,j}$ and $DAct_{i,j}$, respectively. Like before, projection π is extended to sets and functions in a pointwise manner. Thus, the projection $\pi_{(\mathcal{I}\times\mathcal{A}_{i,j})\times\mathcal{A}_{i,j}}(RAct_{i,j})$ of activity $RAct_{i,j}$ is in $((\mathcal{I}\times\mathcal{A}_{i,j})\times\mathcal{A}_{i,j})^{\mathbb{R}_{p+}}$. As a shorthand, we write $pRAct_{i,j}$ for the projected activity. For $DAct_{i,j}$, we write $pDAct_{i,j}$ to denote the activity resulting from adding assignment $rs' = \mathit{false}$ to each of its computation laws and projecting that activity on $(\mathcal{I}\times\mathcal{D}_{i,j})\times\mathcal{D}_{i,j}$. The modification of rs is needed, because we want to consider the activities in isolation. It provides that the reinitialization of an activity caused by a transition in the discrete part $DCom$ ends after an activation of the activity. Within $DAna$ the modification is performed by the last activity of a sequential composition. However, when we regard an activity in isolation, it must be performed by the activity itself.

The space built by the controlled variables not occurring in the definition of $RAct_{i,j}$ or $DAct_{i,j}$, respectively, is denoted by $\overline{\mathcal{A}}_{i,j}$ and $\overline{\mathcal{D}}_{i,j}$, respectively. $\mathcal{A}_{i,j}$ is called the *controlled space* of $RAct_{i,j}$ and $\overline{\mathcal{A}}_{i,j}$ is called its *unconstrained space*, and analogously for the discrete-time case. Note that $\mathcal{A}_{i,j}\times\overline{\mathcal{A}}_{i,j}$ is isomorphic to $\mathcal{S}_C$ and $\mathcal{D}_{i,j}\times\overline{\mathcal{D}}_{i,j}$ is isomorphic to $\mathcal{S}_D$. The above statement that activities are the identity on variables not occurring in their definition can now be formalized by: $\pi_{\mathcal{I}\times\overline{\mathcal{A}}_{i,j}\times\overline{\mathcal{A}}_{i,j}}(RAct_{i,j}) = \mathrm{i}_{\overline{\mathcal{A}}_{i,j}}$ (time-extended additive identity) and similar in the discrete-time case. This is needed in the proof of the following theorem. Finally, we write $\mathcal{O}_{i,j}$ for the factor space corresponding to the output variables occurring in $RAct_{i,j}$. The discrete-time activities have the same output spaces $\mathcal{O}_{i,j}$, because of the demands on the variables occurring in them, as described above. The output relaxation of the relaxed HySChart is denoted by R_{int}.

Under these assumptions the following theorem states that assumption 4 of Theorem 5.1, which basically demands that $DAna$ refines $RAna$, is satisfied if it is satisfied for each activity. (As introduced in Section 5.4.1, we also say that $DAna$ is a sample-and-hold-refinement of $RAna$ if that assumption 4 is satisfied.)

Theorem 5.2 (Compositional activity discretization.) *Provided hierarchic activities are completely independent from each other (see above), the following holds:*

On any interval where DAna is only reset at the beginning, DAna is a sample-and-hold-refinement of RAna for sampling rate T and considered input set $I \subseteq \mathcal{I}^{\mathbb{R}_{p+}}$ (assumption 4 of Theorem 5.1), if the respective property holds for the individual activities. More precisely, this means that on any interval where there is only one reset at the beginning, each activity complemented by the treatment of rs and projected on its controlled space, i.e. $pDAct_{i,j}$, $i \in \{1,\ldots,n\}, j \in \{1,\ldots,m_i\}$, must be a sample-and-hold-refinement of $da(pRAct_{i,j})$, where the output relaxation $pR^{i,j}_{int}$ used for $pRAct_{i,j}$ results from restricting relaxation R_{int} to the output space $\mathcal{O}_{i,j}$. Formally, for every activity $pDAct_{i,j}$ and every interval $\delta = \{k_1+1,\ldots,k_2+1\}$, $k_1 \leq k_2$ in $\mathbb{N}$, the following must hold:

$$
\begin{array}{ll}
\forall \iota \in I.\, \forall \iota' \in \mathcal{I}^{\mathbb{N}}.\, \forall \tau' \in \mathcal{D}^{\{k_1,\ldots,k_2+1\}}_{i,j}. & \\
\quad \iota' \in sample_{\mathcal{I},T}(\iota) \ \wedge & \\
\quad \tau'.rs(k_1) = true \ \wedge & \textit{(reset at beginning)} \\
\quad \forall k \in \delta.\, \tau'.rs(k) = false \ \wedge & \\
\quad (\iota'|_\delta, \tau'^1|_\delta, \tau'|_\delta) \in pDAct_{i,j}|_\delta & \textit{(τ' satisfies $pDAct_{i,j}$)} \\
\quad \Rightarrow & \\
\quad \exists \tau \in \mathcal{A}^{[k_1\cdot T,(k_2+1)\cdot T)}_{i,j} \cap C. & \\
\qquad (\iota|_{[k_1\cdot T,(k_2+1)\cdot T)}, \tau, \tau) \in da(pRAct_{i,j})|_{[k_1\cdot T,(k_2+1)\cdot T)} \ \wedge & \textit{(τ satisfies $pRAct_{i,j}$)} \\
\qquad \forall k \in \{k_1,\ldots,k_2\}.\, \pi_{\mathcal{A}_{i,j}}(\tau'(k)) = \tau(kT) \wedge & \textit{(sampling instants OK)} \\
\qquad \pi_{\mathcal{A}_{i,j}}(\tau'(k_2+1)) = \lim_{x \nearrow (k_2+1)\cdot T} \tau(x) \ \wedge & \textit{(last value OK)} \\
\qquad hold_{\mathcal{O}_{i,j},T}(\pi_{\mathcal{O}_{i,j}}(\tau')) \subseteq pR^{i,j}_{int}(\pi_{\mathcal{O}_{i,j}}(\tau)) & \textit{(hold extension OK)}
\end{array}
$$

where τ'^1 is the right shift of τ' by one time unit and C is the set of continuous functions in $\mathcal{A}^{[k_1\cdot T,(k_2+1)\cdot T)}_{i,j}$.

The inductive proof is rather technical and given in Appendix A.2.2. The theorem allows us to consider the dynamics caused by individual activities in isolation, provided the activities are completely independent, as described above.

5.4.3.2 Using Numerical Techniques for Discretization

One way of obtaining discrete-time activities from continuous-time activities is by using techniques from numerical mathematics. In the following we outline such basic numerical techniques and discuss how they can be employed in our settling.

Numerical techniques. Numerical techniques for ordinary differential equations assume initial value problems of the form $\dot{y} = g(y(t), t)$, $y(t_0) = y_0$ given. They compute approximations y_k to the exact solution $y(kT)$ at point $k \cdot T$, $k \in \mathbb{N}$, for a step size T.[18] In our context g is independent from time, but depending on external input. For input x it can be written as $g(y(t), t) = f(y(t), x(t))$ for a function f. We use this

[18] Techniques with variable step size are not of interest for DiSCharts, which are based on periodical sampling.

notation in the following presentation. Numerical mathematics distinguishes between singlestep and multistep techniques. In singlestep techniques only the last approximation y_k of y is used to compute the next approximation y_{k+1}. Multistep techniques also use older approximating values of y for the computation of the next value. We focus on singlestep techniques here, as they can be understood more intuitively.[19]

Two simple singlestep techniques, which are also used in the context of discrete-time controller design (see below), are the Euler backward method and the trapezoidal method [CG92, Jai79, Oga87]. In the Euler backward method the integral $\int_{kT}^{(k+1)T} f(y(\tau), x(\tau))d\tau$ is approximated by $T \cdot f(y((k+1)T), x((k+1)T))$. This can be regarded as extending the value of f at time $(k+1)T$ backward in time. The resulting equation for y_{k+1} is $y_{k+1} = y_k + T \cdot f(y_{k+1}, x((k+1)T))$, where $x((k+1)T)$ is the value of x at time $(k+1)T$. The values of y_k and y_{k+1} are the computed approximations for y. The computation law for y_{k+1} is obtained by solving the equation for y_{k+1}.

The trapezoidal method approximates the integral $\int_{kT}^{(k+1)T} f(y(\tau), x(\tau))d\tau$ by making a linear interpolation between the values at kT and $(k+1)T$ and by computing the (trapezoidal) area under the interpolating curve. This yields the following equation for y_{k+1}: $y_{k+1} = y_k + \frac{T}{2}(f(y_k, x(kT)) + f(y_{k+1}, x((k+1)T)))$. Solving this equation for y_{k+1} results in the computation law for y_{k+1}. A symbolic solution of the equation exists, if f is a linear function in y. If there is no symbolic solution, the equation must be solved numerically. As Schwarz mentions, the trapezoidal method is usually superior to the Euler method as far as the error is concerned [Sch88].

Further popular single step methods are the Runge-Kutta algorithms. Besides y_k and $x(kT)$, they also use the value of $f(y(t), x(t))$ at further points in the interval $[kT, (k+1)T]$ to compute approximation y_{k+1}. As DiSCharts sample their input, the current external input is not available at such intermediate points. Thus, these methods can only be applied in our context if an activity is independent from external input.

Errors. As far as error estimation is concerned numerical mathematics distinguishes the local discretization error made in one computation step and the global error which accumulated since the start of the algorithm [BSMM97]. For our purpose the global discretization error is primarily interesting. It is defined as $v_k = y(kT) - y_k$, where y_k is computed in k iterations of the algorithm and $y(kT)$ is the value of the exact solution. Local and global error depend on the used algorithm and the underlying problem. For error estimation, the computation law (or function) which defines how y_{k+1} is computed from old approximations and the values of x, is usually required to be Lipschitz continuous. In this case, the global error on an interval $[0, kT]$ is in general bounded by a function which is exponential in k. Hence, in the worst case

[19] Nevertheless multistep techniques can also be applied in DiSCharts. Auxiliary controlled variables can be introduced to store old values of y.

the global error diverges for $k \to \infty$ (cf. [Sch88]). Numerical mathematics therefore introduces the notion of numerical stability for methods for the solution of differential equations. A numerical method is called stable iff the the global error remains bounded for $k \to \infty$ [Jai79].[20] It turns out the stable methods, e.g., exist for differential equations which describe damped exponential behavior. Such behavior often occurs when modeling physical systems and justifies that numerical techniques are well suited for such problems. As an example, we consider the function $y(t) = e^{-ct}$, $c > 0$, which is the solution for initial value problem $\dot{y} = c \cdot y$, $y(0) = 1$. Using the trapezoidal method yields that y_{k+1} is calculated by $y_{k+1} = \frac{1-\frac{1}{2}cT}{1+\frac{1}{2}cT} \cdot y_k$. As the step size T is greater that 0, this implies that the approximated values decrease. The global error is given by $v_k = e^{-c \cdot kT} - (\frac{1-\frac{1}{2}cT}{1+\frac{1}{2}cT})^k$, where we start with $y_0 = 1$. It converges to 0 for $k \to \infty$.

Numerical techniques in DiSCharts. In the context of the discretization of continuous-time activities with numerical techniques, the step size T corresponds to the sampling rate. If numerical techniques are used to find a sample-and-hold-refinement of a relaxed activity, the technique and the sampling rate must be chosen such that the global error for the considered variable y is less or equal to the discretization error permitted by the relaxed activity (Section 5.2.2).

If the value of y is an output of the HySChart, which is to be refined, we also have to regard the distance d_k between the exact value of y on the interval $[kT, (k+1)T)$ and the last approximation y_k. It must be bounded from above by the output relaxation used for y in the relaxed HySChart. Under the assumption that y is Lipschitz continuous with constant l on the regarded interval, d_k is constrained by $d_k \leq v_k + l \cdot T$, where v_k is the global error at time kT.

Thus, using numerical methods for the discretization of the activities in a relaxed HySChart leads to further constraints on the sampling rate T. It must be chosen such that (1) the global error for the numerical methods used as discrete-time activities lies within the relaxation of the exact activities and (2) such that the error between sampling instants for every output variable lies within the output relaxation of the relaxed HySChart. Together with the constraints on T which stem from refining the discrete part (Section 5.4.2), these further constraints yield an upper bound below which $T > 0$ may be selected to obtain a correct refinement of a (relaxed) HySChart.

Syntactically numerical computation laws of the form $y_{k+1} = F(y_k, x((k+1)T), T)$ for a function F are described by their characteristic predicate in DiSCharts, as described in Section 5.3.3. In these predicates we write y' for y_{k+1} (the next value of controlled variable y), y refers to its latched value y_k and $x^\backslash$ stands for the current input value $x((k+1)T)$. If older values of the input or of y are also needed to compute the next value, the activity can use the extended state space which is not relevant for

[20] This notion of stability should not be mixed up with the general notion of stability defined in Section 6.3.3.

the discrete part of the DiSChart, denoted by $\mathcal{S}_{aux}$ in Section 5.3.3, to store such old values and can update them in each activation. For instance, this is necessary for the trapezoidal method, where the old input is needed, and for multistep methods.

If an activity specifies linear evolution of a variable, like the activity for variable *now*, a simple method, like Euler backward, already suffices to compute the *exact* value of it. In fact the discrete-time activity for *now* in Section 5.3.3 corresponds to Euler backward.

5.4.3.3 Using Control Theory Methods for Discretization

Various methods for obtaining an approximating discrete-time control law from a given continuous control law exist. In the following we briefly list such methods. A detailed presentation can, e.g., be found in [Oga87].

Discretization methods. Starting from a differential equation $\dot{y}(t) = f(y(t), x(t))$ describing the continuous-time control law one class of methods from control theory transforms the law to the following form which relates the control law's output y at successive sampling instants: $y((k+1)T) - y(kT) = \int_{kT}^{(k+1)T} f(y(\tau), x(\tau))d\tau$. The integral in the equation is then approximated by simple numerical techniques (Euler backward or trapezoidal method) and the resulting equation defines the approximating discrete-time control law. Some other methods are based on constructing a discrete-time controller which *exactly* matches a continuous-time controller's output at sampling instants when fed with specific input. Considered inputs here usually are impulses or discrete steps. Similarly, there also are methods which do not try to match the exact output, but only some characteristic parameters of the output for certain input. Such parameters, e.g., are the settling time until a steady state[21] is reached after a step-shaped input, or the frequency and amplitude characteristics of the output in response to sinusoidal inputs. Note that the response to step-shaped input and sinusoidal input is also called step response and frequency response, respectively. Such tests inputs and the reaction they cause play an important role in control theory since they often allow the designer to draw conclusions on the general behavior of the regarded system. A further method is based on matching poles and zeros of the Laplace transform of the continuous-time control law. As these parameters greatly influence the controller's frequency response characteristics, this method ensures that the resulting discrete-time control law has a frequency response similar to the original one.

All these methods have in common that the behavior of the resulting discrete-time controller is close to that of the original one w.r.t. some characteristics, but less faithful w.r.t. others. For instance, some methods result in faithful step response, but

[21] A steady state is a state in which a system will remain forever, unless it is perturbed by its environment.

largely different frequency response. Furthermore, the fidelity of all those methods is increased by increasing the sampling frequency. The resulting controllers usually are not acceptable for sampling frequencies below or close to twice the maximum frequency in the input signal, which corresponds to Shannon's theorem (Section 2.1). As Ogata [Oga87] mentions, usually eight to ten times of this frequency is required for sampling to obtain good quality.

For the aim of finding a sample-and-hold-refinement of a relaxed activity *RAct*, the more or less heuristic character of the outlined methods hampers their use for refinement. In contrast to the numerical techniques from above, the accuracy of these methods is not clear. However, we can employ such methods if activity *RAct* expresses exactly such requirements which are satisfied by one of the methods: If *RAct* is specified such that it only reflects an abstract requirement, like a constraint on the settling time in the step response, a control theory method which constructs a discrete-time controller that matches this paremeter may be used. Below we give an example of such an activity. On the level of semantics such activities allow all those trajectories which comply with the required parameters.[22]

Direct methods. The situation is similar if direct methods for the construction of discrete-time controllers are supposed to be applied for controller design. Under specific circumstances such methods, like *quadratic optimal control* [Oga87], are able to directly construct discrete-time control laws for analog plants which satisfy certain requirements. In the case of quadratic optimal control, these requirements are stability of the resulting control system and optimality w.r.t. a cost function which is quadratic in the state of the regarded system. Stability will be introduced in detail in Section 6.3.3. In the context of this section it informally means that small disturbances of a system's state only have a small effect on the system's behavior. As stability practically is required from any control system, we now explain how activities modeling this requirement can be expressed.

We extend activity syntax to also allow us stability constraints of the following form as activities: $a \equiv \vec{u}$ *such that* $\vec{x} = 0$ *is stable for* $\dot{\vec{x}} = f(\vec{x}, \vec{u})$. This activity a is meant to express all evolutions of the controlled variables in tuple $\vec{u}$ such that point $\vec{x} = 0$ is stable under the assumption that the relationship between $\vec{x}$ and $\vec{u}$ is given by $\dot{\vec{x}} = f(\vec{x}, \vec{u})$. $\vec{x}$ is assumed to be in the input space of the considered component and $\vec{u}$ is assumed to be output to the component from which the considered component receives $\vec{x}$. The semantics of this kind of activities is defined by:

$$\begin{aligned}[\![a]\!] \;=\; \{(\iota, \sigma, \sigma) \in \mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}} \;\mid\; \forall \delta.\, \forall t \in \delta. \\ (\tfrac{d}{dt}\iota.\vec{x})(t) = f(\iota.\vec{x}(t), \sigma.\vec{u}(t)) \Rightarrow \mathit{stable}(0, \iota.\vec{x}, \delta)\}\end{aligned}$$

where $\delta \subseteq \mathbb{R}_+$ is an interval, $\omega.\vec{y}$ denotes those components of input or data-state stream ω which correspond to the variable names in tuple $\vec{y}$ and $\frac{d}{dt}\omega.\vec{y}$ is the tuple

[22] An output relaxation R_{int} is also needed for such activities on order to allow the jumps in the discrete-time implementation.

of derivatives of those components of ω which correspond to the variable names in $\vec{y}$. Predicate $stable(0, \chi, \delta)$ is defined as state-based stability of point 0 for trajectory χ w.r.t. a given topology and interval δ, according to Section 6.3.3.2. For the time being, this may be regarded as equivalent to $stable(0, \chi, \delta) \equiv \forall \varepsilon > 0. \exists \nu > 0. d(0, \chi(0)) < \nu \Rightarrow \forall t \in \delta. d(0, \chi(t)) < \varepsilon$, where a metric space with metric d is assumed. State-stream σ in the above definition is arbitrary on intervals where the dynamics of the input is not as expected. On all other intervals, σ is required to evolve in a way which guarantees that trajectory ι evolves in a stable manner. Universal quantification over intervals is needed to ensure that $(\iota, \sigma, \sigma)|_{[u,v)} \in [\![a]\!]|_{[u,v)}$ does not hold for arbitrary streams in which the input dynamics only differs from the expected dynamics on an interval that is not considered in $[u, v)$ anyway. Controlled variables which are not a component of $\vec{u}$ are not constrained by the activity. Note that because of our machine model, the discrete part of a HySChart must also agree on the stream σ selected by the analog part. In particular, it determines the start value of σ.

Control theory techniques in DiSCharts. Obviously, the above activity definition allows refinement with techniques that generate stable discrete-time controllers. We only have to ensure that the resulting discrete-time activity also satisfies the output relaxation specified for the relaxed HySChart, like in the case of numerical methods as discrete-time activities. However, this "stability activity" is just an example. Further requirements, besides stability of the overall system, may involve prescribed accuracy of the controller, fast response to changes and certain behavior w.r.t. periodic inputs.[23] Similar to stability they can as well be given a semantics compatible to activities and thereby also allow the use of existing control theory techniques for controller design.

Thus, control theory techniques can be applied in case of a very loose specification of the analog part, e.g. only reflecting stability requirements and certain loose intervals in which the continuous evolution is required to be. Typically this can occur in the early development steps for the controlling component of a hybrid systems, like e.g. in the controller component of the EHC. There, the restriction of the continuous evolution to certain intervals can be regarded as expressing the limited range the available actuators have for influencing the plant. The need for a loose specification of the analog part suits well to the relaxation of HySCharts which also stresses the need for uncertainty in the analog part.

5.4.4 Time Refinement of the EHC's Controller

In this section we employ the above results to show that the DiSChart given in Section 5.3.6 is a discrete-time refinement of the relaxed HySChart of Section 5.2.5.

[23] Nevertheless activities may only define sets with a certain closure (or admissibility) property. This property is introduced in Definition A.2 of Appendix A.1.1 and called divergence closure.

Example 5.5 The refinement proof is based on Theorem 5.1. For the input on discrete channel *bend* and hybrid channel *fHeight* we use the set of evolution constraints $I \subseteq \mathcal{I}^{\mathbb{R}_{p+}}$ which is defined as follows: For *bend*, we assume that its minimum event separation is given by *bs*. For *fHeight*, we assume that its minimum event separation is given by *fs*, the Lipschitz constant constraining the exact evolution during time periods where the channel's time stamp is constant is *fl* and the error *fe* is associated with the channel. Formally I is given by:

$$\begin{aligned} I \;=\; \{\; (\beta,\varphi) \in (\mathbb{R}_+ \times (\mathbb{R}\times\mathbb{R}_+))^{\mathbb{R}_{p+}} \;\mid\; & \beta \in cs(E) \;\wedge\; mes(\beta) \le bs \;\wedge \\ & \varphi \in cs(\mathbb{R}) \;\wedge \\ & \exists \varphi' \in cs(\mathbb{R}).\, \varphi'.t = \varphi.t \;\wedge\; mes(\varphi') \le fs \;\wedge \\ & \quad Li(\varphi') \le fl \;\wedge\; \forall u.\, |\varphi.val'(u) - \varphi.val(u)| \le fe \;\} \end{aligned}$$

where β is the stream on channel *bend* and φ is the stream on channel *fHeight*. Functions $mes(.)$ and $Li(.)$ are defined in Section 3.4, and $cs(.)$ is defined in Example 5.4. For the evolution of the relaxed HySChart's state no specific constraints are needed for refinement, because the variables in the state space, in particular *aHeight*, do not trigger transitions of the HySChart. (In fact, all transitions in the chart are triggered by input *bend* and *fHeight*.) Thus, we define set S of evolution constraints in the most liberal way by setting $S = (n \cdot \mathcal{S}_C)^{\mathbb{R}_{p+}}$, where $n \cdot \mathcal{S}_C$ is the HySChart's state space.

As start states we consider arbitrary states $s \in n \cdot \mathcal{S}_D$, $s' \in n \cdot \mathcal{S}_C$ with $s.now = 0$, $s.rs = init$ and $s' = \pi_{n\cdot\mathcal{S}_C}(s)$, where $n \cdot \mathcal{S}_D$ is the state space of the DiSChart. The data-state space of the relaxed HySChart $\mathcal{S}_C$ is a factor space of $\mathcal{S}_D$, because the DiSChart uses the same variables as the HySChart plus the further variable rs. Thus, assumption 1 of Theorem 5.1 is satisfied.

It is easy to see that the invariants of the DiSChart (Figure 5.8) result from first deriving the exact invariants of the HySChart in Figure 5.5 and then adding disjunct $now \; mod \; T \neq 0$ to each of them. The actions in the DiSChart coincide with those in the HySChart except for the action body $rs' = true$ which is implicitly added to every action. Thus, due to Section 5.4.2, the discrete part of the DiSChart satisfies assumptions 2 and 3 of Theorem 5.1 if the sampling rate T satisfies the following set of constraints:

$$\{\; T \le \varepsilon_{bend}\,,\; T \le \tfrac{\varepsilon_{i1} - 2\cdot fe}{fl}\,,\; T \le \varepsilon_{fHeight} \;\}$$

These constraints are derived from the relaxed invariants and the evolution constraints on the input in the way described in Section 5.4.2. Note that the invariants of *inTol*, *up* and *down* all result in the same constraint since we selected $\varepsilon_{i1} = \varepsilon_{i2} = \varepsilon_u = \varepsilon_d$ (Section 5.2.5).

In order to satisfy assumption 4 of Theorem 5.1, we regard the individual activities. All activities which reference variable *aHeight* are independent from each other, since they are associated with different nodes. This corresponds to the disjoint sum of the activities on the level of semantics. The implicit activity incrementing *now* is sequentially composed with the other activities (which reflects the hierarchy of nodes), but it does not reference *aHeight*. Thus, the independence of activities as required in

Section 5.4.3.1 is satisfied and Theorem 5.2 can be used. According to that theorem we have to show that on any interval where there is only one reset at the beginning, *a_idle* is a sample-and-hold-refinement of $da(a\_const)$, where only the state space corresponding to the variables *rs* and *aHeight* is considered for *a_idle*, and for *a_const*, only the state space for *aHeight* is considered. Similarly, we have to show that *a_rise* is a sample-and-hold-refinement of *a_inc*, that a_fall is a sample-and-hold-refinement of *a_dec* and that the implicit discrete-time activity incrementing *now* also is a sample-and-hold-refinement of the implicit continuous-time activity for *now*. Here, we only consider the case of *a_rise* and *a_inc*. The others are similar. Informally, we establish that for any evolution of *aHeight* according to discrete-time activity *a_rise*, there is a corresponding evolution of it according to the continuous-time activity *a_inc*. Due to the simple character of *a_rise* and *a_inc*, such a corresponding continuous-time evolution in *a_inc* can be constructed by linear interpolation of the discrete-time evolution in *a_rise*. Formally, we proceed as follows. Let $\iota \in I$ be an input and define $\iota' \in sample_{\mathcal{I},T}(\iota)$. Let $\tau' \in \mathcal{D}^{\{k_1,\ldots,k_2+1\}}$, where $\mathcal{D}$ is the controlled space of *a_rise*, corresponding to variables *rs* and *aHeight*. Furthermore, let τ' be such that it only contains a reset at time k_1 and satisfies *a_rise* on the interval $\delta = \{k_1+1,\ldots,k_2+1\}$, $k_1 \leq k_2$ in $\mathbb{N}$ (see Theorem 5.2 for the formal definition of these notions). By definition of *a_rise* this implies that for all $k \in \delta$, $\tau'.aHeight(k) = \tau'.aHeight(k-1) + c_k T$ holds for a $c_k \in [cp_-, cp_+]$. We use linear interpolation to define the continuous $\tau \in \mathcal{A}^{[k_1 \cdot T,(k_2+1)\cdot T)}$, where $\mathcal{A}$ is the controlled space of *a_inc* corresponding to variable *aHeight*: For all $k \in \delta$ and $t \in [(k-1)T, kT)$ we define $\tau.aHeight(t) = \tau'.aHeight(k-1)+c_k \cdot (t-(k-1)T)$. By definition τ satisfies $da(a\_inc)$ projected on $\mathcal{A}$ (note that $da(.)$ allows that τ is not smooth). Furthermore, $\tau.aHeight$ and $\tau'.aHeight$ obviously agree at sampling instants and in their "last value". It remains to show, that the *hold* extension of τ' is within the output relaxation of τ, i.e. they may be no further than $\varepsilon_{int}.aHeight$ apart ($\varepsilon_{int}.aHeight$ was introduced in Section 5.2.5 for the relaxed HySChart). By definition of τ the distance between $\tau.aHeight(t)$ and the hold extension of $\tau'.aHeight$ is less than $c_k T$ for $t \in [(k-1)T, kT)$, $k \in \delta$. Assuming the worst case, this results in the further constraint $T \leq \frac{\varepsilon_{int}.aHeight}{cp_+}$ on the sampling rate.

Considering refinement of the other activities we obtain similar constraints. For a_fall and *a_dec* we get constraint $T \leq \frac{\varepsilon_{int}.aHeight}{|ev_-|}$. In the case of *a_idle* and *a_const* the deviation between the *hold* extension of the discrete-time evolution and the continuous-time evolution between sampling instants is equal to the deviation at sampling instants, since *aHeight* is constant for these activities. Thus, it is zero (see Section 5.3.6). For the implicit activities defining the evolution of variable *now*, $now' = now + T$ and $\frac{d}{dt} now = 1$, respectively, no output relaxation has to be considered, because *now* is not an output of the HySChart and the DiSChart. Hence, we get no further constraints on T.

Provided T satisfies the above two constraints derived from the activities, Theorem 5.2 discharges assumption 4 of Theorem 5.1. Moreover, assumption 5 of Theorem 5.1 is trivially satisfied, because S does not constraint the evolution of the HySChart's state.

As a result, Theorem 5.1 yields that the DiSChart given in Section 5.3.6 is a discrete-time refinement of the relaxed HySChart of Section 5.2.5, if T is chosen such that the following constraint is satisfied:

$$T \leq min\{\varepsilon_{bend}, \frac{\varepsilon_{i1}-2\cdot fe}{fl}, \varepsilon_{fHeight}, \frac{\varepsilon_{int}.aHeight}{cp_+}, \frac{\varepsilon_{int}.aHeight}{|ev_-|}\}$$

As a quantitative example, for $\varepsilon_{bend} = 120\ ms$, $\varepsilon_{i1} = 2\ mm$, $fe = 0.5\ mm$, $fl = \frac{1}{100}\frac{mm}{ms}$, $\varepsilon_{fHeight} = 200\ ms$, $\varepsilon_{int}.aHeight = 0.1\ mm$ and $cp_+ = |ev_-| = \frac{1}{1000}\frac{mm}{ms}$ a sampling rate $T = 100\ ms$ suffices.[24] □

5.4.5 Sampling Rate Validation

As Example 5.5 shows, the theorems from above can be used to derive a sampling implementation from a relaxed HySChart. However, methodologically they can also be used in the other direction, in order to validate that a given sampling rate for a system is adequate, i.e. that it results in an acceptable relaxation of an exact HySChart. In a development process, this is useful in cases where the sampling rate is already fixed, because, e.g., some legacy hardware is supposed to be used. In this section we briefly outline how sampling rate validation can be performed with the above methods.

We assume a DiSChart and an (exact) HySChart with exact invariants given. Both must have the same structure and the same actions. For the analog parts we assume that the discretization error between the discrete-time analog part and the continuous-time analog part at sampling instants is zero. Furthermore, we require that the invariants in the DiSChart correspond to those in the HySChart, but with additional conjunct $now\ mod\ T \neq 0$. Using the techniques from Sections 5.2.1 and 5.2.2 we construct a relaxed HySChart from the exact HySChart. For the bounds ε occurring in the relaxation of invariants we use free variables. The relaxed analog part is defined to be equal to the exact analog part (motivated by the lack of a discretization error). Finally, for the constants used in the output relaxation further free variables are introduced.

The aim now is to determine bounds for the variables used in the relaxations such that Theorem 5.1 yields that the DiSCharts refines the relaxed HySChart. In order to satisfy the conditions on the discrete parts which are required in Theorem 5.1, we can use the method described in Section 5.4.2 to compute constrains involving the sampling rate, the variables used in the relaxation of the invariants and Lipschitz constants and error bounds limiting the continuous behavior of the input and the analog part. These Lipschitz constants and error bounds are required to be given correctly. Moreover, to satisfy the condition on the analog parts which is required in Theorem 5.1, it suffices to consider the output relaxation, since the discretization error is assumed to be zero. For the satisfaction of this condition, constraints w.r.t. the

[24] mm denotes millimeters and ms denotes milliseconds.

variables used in the output relaxation can be derived from the Lipschitz constants and the sampling rate, as indicated at the end of Section 5.4.3.2 in the context of numerical techniques. When solving all the constructed constraints for the free variables used in the relaxations, they provide lower bounds on these variables. The bounds denote that for lower values of the relaxation variables, the DiSChart is not guaranteed to be a refinement of the relaxed HySChart. In other words, the designer must expect delays in the reaction to events, errors in the detection of threshold crossings and intersample errors between the DiSChart and the exact HySChart which are between zero and the computed bounds. Based on these bounds, a designer can decide whether the given sampling rate is appropriate for the developed model of the system w.r.t. reaction to events, threshold crossings and intersample behavior, or whether it has to be modified to achieve higher accuracy. Remember, however, that this decision is based in the optimistic assumption that the discretization error is zero.

5.5 Time Refinement in Component Networks

As we have seen in Section 5.4.2 finding a DiSChart whose extension by *sample* and *hold* is a refinement of a given relaxed HySChart usually requires various knowledge about the HySChart's environment. For instance, for continuous and hybrid input channels Lipschitz constants and error bounds constraining their continuous evolution periods are typically required. If the actual input does not satisfy the input constraints which are used in an application of Theorem 5.1 to proof that a DiSChart refines a HySChart, the refinement relation need not hold. Thus, if a network of components (described by a HyAChart) is supposed to be refined by discretizing a component in it on basis of Theorem 5.1, we must ensure that the network behaves in a way which guarantees that the assumptions on the component's input are satisfied.

Assumption/commitment reasoning. In principle this requires an assumption/commitment methodology [SDW95]. The situation is uncritical, if the component network is such that the assumptions on the input used for the refinement of a component are satisfied independently from the component's output. However, if the assumptions on the component's input in the network only hold if the component's output satisfies some commitments , i.e. feedback is involved, care is necessary to avoid circular reasoning. [SDW95] identifies conditions under which such circular reasoning may be used in an untimed formalism. When we informally adapt these conditions to our continuous-time framework they include that (1) there must be a finitely large delay with which the input is affected by the regarded component's output and (2) the assumptions must define a *divergence closed* set. Divergence closure is similar to admissibility in discrete time. For instance, assumptions which only regard the input on finite time intervals define divergence closed sets. A formal definition is given in Definition A.2 of Appendix A.1.1. The kinds of constraints we used in Section 5.4 all result in divergence closed sets. We do not regard assumption/commitment reasoning

further in this thesis, but we congest that the results from [SDW95] can be carried over to the continuous-time case.

Note that provided the assumptions on the input hold, compositionality of multiplicative hierarchic graphs w.r.t. refinement guarantees that the refinement of one component in a HyAChart results in a refined HyAChart.

Justification of input constraints. One source of characteristics like the minimum event separation or Lipschitz constants of the input a controller receives from its environment are physical limitations in the environment. For instance, for the cruise control system of a car, the possible accelerations and deaccelerations are (among others factors) limited by the motor's power and by friction. As such limitations are independent from the output the controller sends to the physical environment, they can conveniently be used as input constraints for the discretization of HySCharts with Theorem 5.1.

Combining discrete-time components. Once we have ensured that the input of a component *Cmp* in a HyAChart satisfies the assumptions under which *Cmp* is refined by a discrete-time component *DCmp* with sampling rate T, we can directly replace *Cmp* by the discrete-time component, embedded between *sample* and *hold* components. More formally, *Cmp* may be replaced by $sample_{\mathcal{I},T} ;_\times DCmp ;_\times hold_{\mathcal{O},T}$ in the HyAChart. Due to compositionality of the multiplicative graph operators, the modified HyAChart refines the original one (Section 3.3.5).

If this is done for a larger number of components in a HyAChart, the refined HyAChart may involve parts in which discrete-time components embedded within *sample* and *hold* components directly communicate with each over continuous-time channels. Provided these discrete-time components are active at the same points in time[25], it would obviously suffice if they communicated over discrete-time channels without *sample* and *hold* components between them. This would enable more efficient simulation and it would allow us to use existing techniques for the refinement of discrete-time component networks in further development steps. We therefore elaborate this idea. First, discrete-time components and communication over discrete-time channels require a different kind of architecture diagrams which we call *DiACharts*. Just as HyACharts, DiACharts are hierarchic graphs that are interpreted multiplicatively. The only difference is that we do not use time model $\mathbb{R}_+$ and dense streams for them, but time model $\mathbb{N}$ and discrete-time streams. The necessary semantic foundation was already introduced in Section 3.3.2, which defined the multiplicative interpretation of hierarchic graphs for different kinds of time models. Note that discrete-time block diagrams can be integrated easily into DiACharts, just as continuous-time block diagrams suit well to HyACharts.

[25] This can often be achieved by modifying some components such that they operate at a higher sampling frequency than actually needed.

Second, we introduce three equalities which can be used for simplifications. They allow us to compose discrete-time components embedded in HyACharts with DiACharts, as desired.

Sequential composition. The continuous-time sequential composition of two sample-and-hold extensions of discrete-time components A and B is equal to the sample-and-hold extension of the (discrete-time) sequential composition of A and B, provided both components use the same sampling rate T:

$$(sample_{\mathcal{I}_1,T} ;_\times A ;_\times hold_{\mathcal{O}_1,T}) ;_\times (sample_{\mathcal{O}_1,T} ;_\times B ;_\times hold_{\mathcal{O}_2,T}) = (sample_{\mathcal{I}_1,T} ;_\times A ;_\times B ;_\times hold_{\mathcal{O}_2,T})$$

where $\mathcal{I}_1$ and $\mathcal{O}_1$ is the input and output interface type, respectively, of A and similarly $\mathcal{I}_2$ and $\mathcal{O}_2$ define the interface for B, with $\mathcal{O}_1 = \mathcal{I}_2$. The equality is graphically depicted in Figure 5.10, top. In the figure, *sample* is abbreviated by s and *hold* by h. This equality holds, because $hold_{\mathcal{O}_1,T} ;_\times sample_{\mathcal{O}_1,T}$ is the identity on $\mathcal{O}_1$ in the discrete time model.

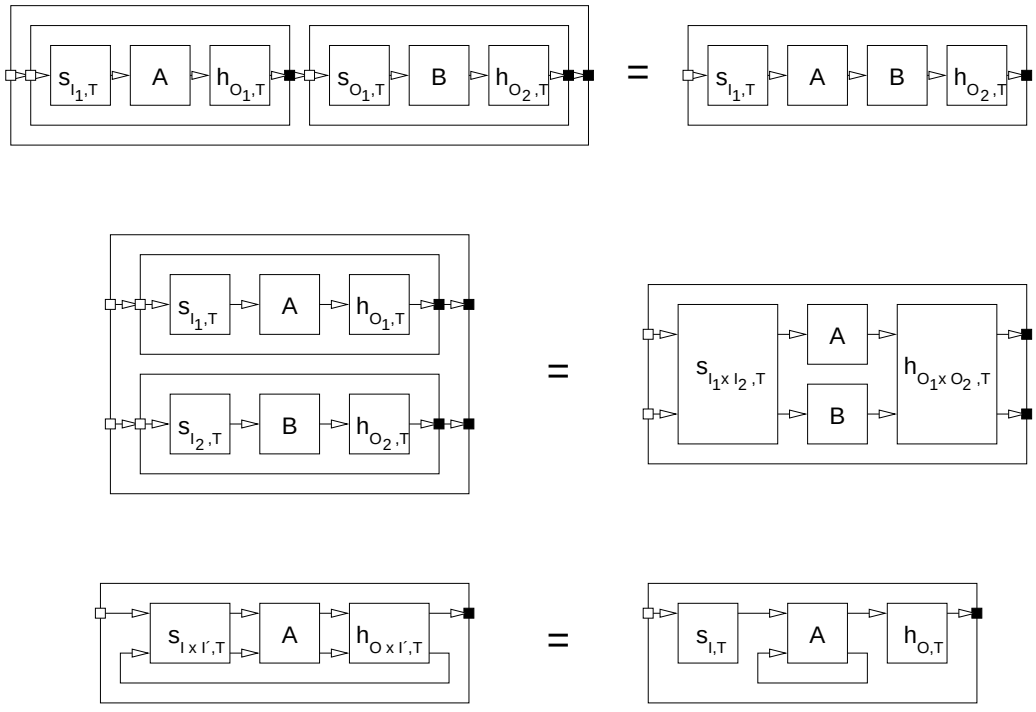


Figure 5.10: Composition of discrete-time components within HyACharts.

Multiplicative visual attachment. The continuous-time multiplicative visual attachment (or parallel composition) of two sample-and-hold extensions of discrete-time components A and B is equal to the sample-and-hold extension of the

(discrete-time) multiplicative visual attachment of A and B, provided both components use the same sampling rate T:

$$(sample_{\mathcal{I}_1,T} \,;_\times A \,;_\times hold_{\mathcal{O}_1,T}) \times (sample_{\mathcal{O}_1,T} \,;_\times B \,;_\times hold_{\mathcal{O}_2,T}) = \\ (sample_{\mathcal{I}_1\times\mathcal{I}_2,T} \,;_\times (A \times B) \,;_\times hold_{\mathcal{O}_1\times\mathcal{O}_2,T})$$

where $\mathcal{I}_i$ and $\mathcal{O}_i$ are the input and output interface types of A and B. The equality is graphically depicted in Figure 5.10, middle.

Feedback. The continuous-time feedback of a sample-and-hold extension of a discrete-time component A is equal to the sample-and-hold extension of the (discrete-time) feedback of A:

$$(sample_{\mathcal{I}\times\mathcal{I}',T} \,;_\times A \,;_\times hold_{\mathcal{O}\times\mathcal{I}',T}) \uparrow_\times^{\mathcal{I}'} = (sample_{\mathcal{I},T} \,;_\times (A \uparrow_\times^{\mathcal{I}'}) \,;_\times hold_{\mathcal{O},T})$$

where the interface types of A are $\mathcal{I} \times \mathcal{I}'$ (input) and $\mathcal{O} \times \mathcal{I}'$ (output). The equality is graphically depicted in Figure 5.10, bottom. It can be proven directly or by applying the transformation rules for the underlying algebra for hierarchic graphs given in [Ste94].

Together with associativity of the operators in multiplicative hierarchic graphs, these equalities can be used to group discrete-time, continuous-time and remaining hybrid components together. This is useful if a separate development process is intended to be pursued for subsystems that are supposed to be implemented in analog, digital or mixed-signal hardware.

5.6 Separate Implementation of the Discrete Part

As a side remark we consider the separate discretization of a relaxed HySChart's discrete part here. In cases where there are high accuracy or timing demands on the analog part, or the analog part is a part of the physical environment, an implementation of a HySChart in a mixed continuous-time/discrete-time manner may be desirable. In this section we therefore explain how a discrete-time implementation of the discrete part of a relaxed HySChart can be obtained and how it is coupled with the analog part which is still based on continuous-time. The output relaxation of the relaxed HySChart is not needed; it may be the identity relation. We only argue informally in the following and will not prove the refinement relation.

In order to derive a discrete-time refinement, denoted by $DCom$, from the discrete part of a relaxed HySChart, the method given in Section 5.4.2 can be applied. It yields $DCom$ and a set of constraints on the sampling rate T. Provided the constraints are satisfied, $DCom$ is guaranteed to refine the relaxed discrete part, denoted by $RCom$, at sampling instants and $RCom$ can remain idle throughout a sampling interval if $DCom$ was idle at the beginning (assumptions 2 and 3 of Theorem 5.1). Furthermore,

for the relaxed analog part assumption 5 of Theorem 5.1 must be valid. This means that the assumptions on its behavior, which are needed to ensure that *RCom* can idle during sampling intervals (assumption 3 of Theorem 5.1), must be satisfied.

We then embed *DCom* in the hybrid machine model of the relaxed HySChart (see Figure 5.4). This embedding is done by adding an interface around *DCom* which samples the discrete part's input and holds its output constant between sampling points. To avoid that the sampled discrete part stops the continuous evolution in the analog part, component $merge_T$ provides that the discrete part's output only interferes with the analog part at sampling points. The $merge_T$ component is defined as follows:

$$\begin{array}{lcl} merge_T & \in & (n \cdot \mathcal{S})^{\mathbb{R}_{p+}} \times (n \cdot \mathcal{S})^{\mathbb{R}_{p+}} \to \mathcal{P}((n \cdot \mathcal{S})^{\mathbb{R}_{p+}}) \\ merge_T(\sigma, \tau)(t) & = & \begin{cases} \sigma(t) & \text{if } t \ mod \ T = 0 \\ \tau(t) & \text{otherwise} \end{cases} \end{array}$$

At sampling instants its output is the input received on the left input channel, and between sampling instants its output is the input received on the right input channel. Figure 5.11 depicts the resulting multiplicative hierarchic graph which replaces *RCom* in the machine model of Figure 5.4. The channels between $sample_{\mathcal{I} \times n \cdot \mathcal{S},T}$, *DCom* and $hold_{n \cdot \mathcal{S},T}$ transmit discrete-time streams. To prove that the component

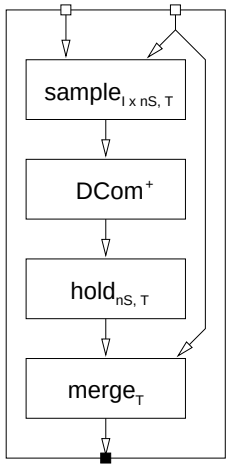


Figure 5.11: Discrete-time implementation of *RCom*.

obtained by this replacement constitutes a refinement of the original component, it suffices to regard the adaptation of *DCom* (Figure 5.11) and *RCom*, since multiplicative hierarchic graphs are compositional w.r.t. refinement. Formally, $(\mathsf{I}_{\mathcal{I}} \times \wp_2^{n \cdot \mathcal{S}}) ;_\times ((sample_{\mathcal{I} \times n \cdot \mathcal{S},T} ;_\times DCom^\dagger ;_\times hold_{n \cdot \mathcal{S},T}) \times \mathsf{I}_{n \cdot \mathcal{S}}) ;_\times merge_T \subseteq RCom$ must hold. Note that the expression between *sample* and *hold* is interpreted w.r.t. the discrete time model $\mathbb{N}$, whereas the underlying time model for the other operators in the formula is $\mathbb{R}_+$. Obviously, assumptions 2 and 3 of Theorem 5.1 help to establish this claim. The theorem itself cannot be applied, because it relies on the discrete-time

machine model for DiSCharts and not on the mixed machine model introduced in this section. We do not go into further details here.

5.7 Discussion and Further Work

5.7.1 Contribution

First, this chapter discussed effects arising in a discrete-time implementation of hybrid systems' components which are specified with an underlying continuous time model. Based on these effects methods were developed which allow us to systematically derive a discrete-time implementation for a component which is specified by a *relaxed* HySChart. The resulting implementation is a refinement of the relaxed HySChart, which guarantees that vital properties of the HySCharts are maintained (see also Section 6.3.6.3). As prerequisite, relaxed HySCharts were introduced as a variant of HySCharts which contains a controllable degree of nondeterminism w.r.t. the time when transitions are taken and w.r.t. to the accuracy of the continuous dynamics (*activities*) and the HySChart's output. Furthermore, a discrete-time dialect of HySCharts, *DiSCharts*, was defined for the specification of the behavior of discrete-time components. DiSCharts may interact with continuous-time components by sampling their input and holding theit output constant between sampling instants. As an analogon to continuous-time activities, activities in DiSCharts specify the evolution of controlled variables at those sampling points where no transition is taken. Description techniques for discrete-time control laws, like discrete-time block diagrams or z-transforms [Vac95], can in principle be used to define these activities.

The discretization methods we introduced require various knowledge about the dynamics of the input a component receives. The minimum time between events, Lipschitz constants and possible errors assiated with continuous periods of evolution are typically needed. For the discretization of a HySChart's analog part we explained how methods from numerical mathematics and control theory can be employed. The method to derive constraints on the sampling rate from the discrete part of a HySChart can also be applied the other way round. For instance, it can be used to compute bounds on possible errors for a given sampling rate, or to compute constraints on Lipschitz constants for which a given error bound is not exceeded.

Although the presentation given here focuses on HySChart, we think that the results can be carried over to other hybrid description techniques as well. In particular, the underlying ideas used to find a discrete-time implementation for a HySChart's discrete part can also be used in the context of other hierarchic state transition diagrams for hybrid systems.

Finally, the chapter explained how the DiSCharts which result from the application of the discretization methods can be integrated into components networks, as described

by HyACharts.

Automation. Automation of the introduced discretization methods is possible by part. Provided a designer specifies which degree of accuracy is desired for the component under development, the invariants and the relaxations of the activities and of the output of a relaxed HySChart can be constructed by a tool from the (incomplete) HySChart containing actions and activities. This proceeds via constructing a HySChart with exact invariants first and then relaxing them. Furthermore, if the designer defines all necessary assumptions on the dynamics of a component's input and on the continuous evolution of its state, the discrete part of a DiSChart, which refines the given relaxed HySChart, and constraints on its sampling rate can automatically be derived. For the analog part the situation is more difficult. However, if the analog part only defines linear functions, a discretization is straightforward (Section 5.4.4). Furthermore, if activities formalize abstract requirements, like stabilization of an assumed environment, automatic techniques from control theory may be applicable, as discussed in Section 5.4.3.3. Numerical techniques are helpful in the discretization of activities, if given continuous evolution is supposed to be approximated with prescribed precision, e.g. with the aim of performing simulations of a plant model.

Guidelines. It is important to note that our method for the discrete-time refinement of a HySChart can only be applied to relaxed HySCharts. In other words, a HySChart can hardly be refined, if it requires the immediate taking of transitions or exact continuous dynamics. As such requirements cannot be satisfied in an implementation anyway, they should be avoided in HySCharts in order to enable flexible refinement. With respect to the development methodology we recommend to specify activities predominantly by liberal constraints in early development steps, such as this is done in the EHC example to model the effects of the actuators (Section 5.2.5). Nevertheless, in some cases sharp activities may also be necessary in early development phases, for instance to express that some system variables follow specific trajectories. However, also in such cases the designer usually is not interested in ensuring the *exact* correspondence to the given trajectory, but the intention rather is to specify that the evolution is *like* the given trajectory. Thus, an acceptable deviation from the ideal case can be specified.

The relaxations used in HySCharts may complicate to establish properties of the overall system, because in the worst case errors may propagate disadvantageously between system components. To avoid such propagation it therefore is advisable to employ discretization of (parts of) a model before the model is split into very fine grained subcomponents. In practice a trade-off is necessary here, because on the other hand too early discretization is contrary to the aim of extensively validating an abstract model before decisions are made which may be difficult to alter later on, like the discretization.

The methods presented here support to analytically derive a sampling rate such that the resulting discrete-time component refines a given hybrid component, which is

specified based on a continuous model of time. An alternative to this analytical way is to use simulation and try to find a suitable sampling rate by trial and error. While this is straightforward, it has not the character of a mathematical proof, like our methods. In practice one has to decide which level of mathematical rigor is needed and reasonable for the system under development. In case of complex discrete dynamics, trial and error methods have severe limitations, since it will usually be impossible to explore all execution modes of a complex system. Nevertheless simulation if highly valuable in order to become familiar with the intended system.

Further applications. A further application of relaxed HySCharts and the refinement technique of Theorem 5.1 is for numerical simulation of hybrid systems. The idea here is that if a designer speficies a relaxed HySChart, simulation techniques (in particular the step sizes of integration algorithms) can be chosen such that the simulation output is a possible output of the specified (relaxed) system. Modular simulation would benefit from the simple sample-and-hold scheme that can be used in the communication of refined components. However, in practice this requires an *automated* way to estimate the error of a numerical method. [AGH+00] identify modular simulation of hybrid sytstems as an aim of future work. Due to the lack of modular simulation with clearly defined accuracy, the simulation semantics for the mixed-signal specification language VHDL-AMS [CB99] is based on the *global* coupling of a numerical solver with discete-event simulation, instead of the *componentwise* coupling of numerical simulation with discrete-event simulation [HSTV00].

Note that the relaxation of a HySChart can also be used to comprise quantization effects. These effects result when a measured continuous quantitiy is transformed to the machine numbers used within a digital machine. They are a further source of inaccuracy in a digital implementation.

5.7.2 Related Work

Refinement. Refinement is a common topic in computer science [Hoa86, Mai87, AL91, Bro93]. An overview of notions of behavior refinement can be found in [vdB00]. The notion of refinement we employ is taken from [Bro97b, Bro99b, BS01].

Refinement of hybrid systems has been considered in [RS98] where the continuous dynamics expressed with so called *(hybrid) action systems* is refined. The refinement of a differential equation by a discrete assignment is also regarded there, but the given rule only compares the start states and end states of the dynamics given by the differential equation and the assigment. The intermediate continuous behavior is ignored.

Refinement of hybrid systems without changing the time model is also regarded in [HMP93]. The basic refinement step there is to replace an interval temporal logic

formula in a so called *abstract phase transition system* by phase transitions. Unfortunately, in [HMP93] the controlling system and its environment are regarded as a single unit, specified by an abstract phase transition system. The lack of an isolated view on system and environment complicates to distinguish these entities. This is disadvantageous, since in a refinement of the system no environment constraints may be added which in turn necessitates a distinction.

Refinement of hybrid systems in the context of the notations *Charon* and *Masaccio*, which are close to HyCharts, is considered in [AGLS01] and [HMP01], respectively. Alur et al. express the compositionality w.r.t. refinement of their formalisms for component composition and for state machines in the form of refinement rules [AGLS01]. For component composition this corresponds to the compositionality of multiplicative hierarchic graphs and HyACharts (Section 3.3.5). For state machines this is similar to the hierarchic refinement principle outlined in Section 4.3 for HySCharts, but more elegant as discussed in that section. A further theorem in [AGLS01], called *context compositionality*, corresponds largely to compositionality of additive hierarchic graphs w.r.t. refinement, because it can be used for the refinement of actions and activities. Time-discretization is not considered in [AGLS01].

[HMP01] states a compositionality result for Masaccio and introduces an assumption/commitment proof principle for this formalism. The compositionality result corresponds to the compositionality of multiplicative and additive hierarchic graphs (Section 3.3.5). The assumption/commitment principle only regards safety properties and excludes circular dependencies not involving a delay. In this respect it is related to the assumption/commitment reasoning in [SDW95] which was discussed in Section 5.5. Interestingly, [HMP01] demonstrated the proof principle along an example where a hybrid model with liberal constraints is refined to an implementation based on sampling. However, the time model itself is not changed and the step to the sampling implementation is not systemized as in this chapter.

Methodology. The methodology of explicitly associating uncertainty and accuracy requirements with certain quantities in a model corresponds well to the Four Variable Model and SCR [Hei96]. There a system specification is given in an ideal manner, corresponding to the (ideal) action guards in HySCharts, and accuracy requirements are given in an additional second step, similar to relaxed invariants in HySCharts. Checking whether a given implementation satisfies the specification amounts to checking whether the behavior of the implementation conforms to the ideal behavior w.r.t. the required accuracy [Hei96]. Moreover, note that the constraints associated with measured quantities in SCR correspond well to the evolution constraints we considered here (see the example in [EKM+93]).

Relaxed models. Henzinger et al. also noticed the too sharp semantics of hybrid automata and defined robust hybrid automata [GHJ97]. For a robust hybrid automaton its set of accepted trajectories (or its behavior, in our sense) does not consist of isolated trajectories but of *tubes* of trajectories. A tube is a set of trajectories which

is topologically open w.r.t. a given metric on the trajectories. Such a tube is accepted by a robust hybrid automaton iff the exact hybrid automaton accepts a set of trajectories which is topologically *dense* in the tube. This means that the tube is accepted, if in the neighborhood of any trajectory in the tube there is a trajectory which is accepted by the exact hybrid automaton. The hope that decidability of the reachability problem, which asks whether certain states of an automaton can be reached, would be improved for the robust version has been disproved in [HR00]. In comparison to relaxed HySCharts, the relaxation leading to robust hybrid automata can be regarded as introducing infinitely small deviations from the exact model. In contrast, the relaxation we use relies on finitely small deviations.

A further relaxation of hybrid automata is introduced by Fränzle [Frä99]. Similar to our relaxation of activities and invariants, these disturbed hybrid automata result from allowing deviations by a given finitely small constant ε from the exact activities and invariants. Based on this idea of deviations the author proves that the reachability problem for disturbed hybrid automata is decidable under certain conditions. These conditions basically mean that the set whose reachability is regarded and the automaton have a certain boundedness property. The problem of finding a discrete time implementation for disturbed hybrid automata is not considered in [Frä99]. A combination of the results of this chapter with the ideas in [Frä99] seems to be a good starting point for further work on verification techniques for relaxed HySCharts.

Delayable transitions. The main motivation for invariants in HySCharts is the ability to express that an enabled transition is not taken immediately. A different approach is taken in the so called *phase transition systems* introduced in [dAM95]. These phase transition systems allow the designer to model delayable transitions by providing a minimum and maximum delay for each transition. A transition can be taken if it has been continuously enabled for its minimum delay and it must be taken if it has been continuously enabled for its maximum delay. While this concept is adequate for real-time systems without analog dynamics and with discrete, message based communication, we think that for hybrid systems, the permitted minimum and maximum delay of transitions is a derived concept, resulting from the desired properties of the system under development and the dynamics of the underlying physical system. For instance, one desired property often is that some continuous variable always remains within given bounds. The bounds together with the variable's possible rate of change imply the permitted maximum delay with which a system has to react when the variable's value comes close to the given bounds. In our opinion it is therefore more natural to think about permitted deviations in some variables' values first ("how close to the bound may the variable get"), before fixing permitted time delays. A way of expressing these deviations is via invariants which overlap with the transition guards, which is done for HySCharts.

From continuous-time to discrete-time. [HG96] analyzes the relationship between a sampling semantics of Duration Calculus and its usual semantics based on

dense time [CHR92]. In particular, the authors also give proof rules which allow them to deduce the validity of duration calculus formulas in discrete-time from the validity of the formulas in continuous-time. This reasoning requires that the sampling period is chosen such that the minimal duration for which a proposition is true is at least as long as the sampling period. This constraint corresponds to the constraints on the sampling period which we derive from the minimum event separation for discrete and hybrid input channels (Section 5.4.2). Analog dynamics are not considered in the paper; it focuses on the real-time aspects.

Discretization. Discretization with the purpose of formal verification is examined in a variety of papers on hybrid systems. Usually, the aim in this field is to find a discrete representation of a hybrid verification problem which can be efficiently represented in a computer and manipulated by it. For example, [Hen91] examines which sets of dense time traces, i.e. which continuous properties, can be described with sets of discrete time traces. A common foundation for model checking of hybrid systems is the representation of the reached state set of a (linear) hybrid automaton by polyhedra in $\mathbb{Q}^n$, where $\mathbb{Q}$ denotes the rational numbers [ACH+95]. The rational numers can be represented by fractions of integers for computer based manipulation. Similarly, [GPV94, Bey01] introduce techniques for timed automata which allow them to only deal with clocks with integer values. In our context of seeking a discrete-time refinement of a given hybrid system, this work can rather be seen as seeking an abstraction of a hybrid system which can be analyzed in a discrete-event manner. In particular, this means that in the cited work transitions, triggered by events, can always be taken immediately when the event occurs.

As we have already outlined in Section 5.4.3.3 a huge amount of work on designing discrete-time controllers for analog systems exists in the field of control theory (see e.g. [Vac95, Oga87]). Furthermore, digital signal processing (DSP) is also concerned with the discretization of analog signals, their digital processing and the reconstruction of analog signals from digital signals [Smi97]. However, to the authors knowledge there is no work in these domains which explicitly discusses how discrete-time implementations of *hybrid* systems are obtained.

Besides the simple numerical methods we regarded in Section 5.4.3.2 there also exists specific work on the numerical simulation of hybrid systems, see e.g. [SBS97] and the overview given in [Mos99]. A key problem in the numerical simulation of hybrid systems is to detect threshold crossings, which cause discrete changes in the hybrid system, with high accuracy and still keep the computational effort low. Usually, this requires the sophisticated adaptation of the step size a numerical solver uses. Due to this variable step size, such techniques can rather be regarded as constructing a related discrete-event system for a given hybrid system than as constructing a related discrete-time system. For the discrete-time implementation of a hybrid system such techniques can not immediately be applied, since step sizes cannot be reduced dynamically here. However, their application for finding a discrete-time refinement of a hybrid system

may be possible in cases where bounds on their worst case error can be given and the algorithms are sufficiently simple to be executed in real-time. The major problem probably is to control the error bounds of the numerical method in a way such that the refinement relation w.r.t. a relaxed hybrid system is satisfied.

5.7.3 Further Work

A primary starting point for further work is to examine when properties of an exact model can be transformed to relaxed properties which hold for a relaxed model. As indicated in Section 5.1 this is not possible in general. Thus, future research should try to identify classes of properties and relaxations for which a transfer is possible. However, due to the discrete switching effects in hybrid systems this probably is extremly difficult to achieve (see also Section 5.1).

Furthermore, the embedding of the refinement techniques for HySCharts, which were introduced in this chapter, into an assumption/commitment methodology as sketched in Section 5.5 should be formalized. In this context, further methods to group discrete-time components within a HyAChart together should also be regarded. This can increase the accuracy of the refinement w.r.t. an exact model, just as in the machine model for DiSCharts which was designed such that intersample errors need not be considered within the component.

Chapter 6

Properties of Hybrid Systems

Regarding the work on hybrid systems one notices that properties of such systems which are examined in case studies often reflect the background of the research groups working on them. Computer scientists often focus on safety properties, such as, for instance, that a certain set of states and certain variable domains are never left. People from control theory often put their emphasis on stability properties. While these distinct focuses are sometimes due to the specific characteristics of the regarded systems, they often also result from a lack of knowledge of the respective other domain. To alleviate this shortcoming we define and classify a set of important properties of control systems within a general framework which should be familiar for computer scientists. With the continuing integration of software and its physical environment in many systems we conjecture that properties which have only been of interest for continuous systems so far will also become important for the software part of embedded systems, which stresses the need for proof methods. For the properties of *stability* and *attraction* we identify topologies where stability is a safety property and attraction is a persistence property in computer science terminology [CMP91].

The properties this chapter considers result from the evaluation of nine hybrid systems case studies and a number of textbooks on control theory. The classification of the properties and the case study evaluation serve as reference for judging the utility of a refinement notion based on trace inclusion as used in the preceding chapters. The result here is that the essential classes of properties are preserved by this refinement notion.

Finally, the chapter proposes two general proof methods for stability and attraction together with some specializations. The methods result from adapting Liapunov-like proof methods from general systems theory to our framework [MT75]. When developing specializations of the methods, parallels to abstraction in computer science and to Galois connections in particular are outlined. The proof methods are applied to two examples, the *electronic height control system* we already introduced in Chapter 3 and a purely discrete self-stabilizing algorithm [Dij82], which is interesting in this

context, because it shows the relevance of attraction also for computer science. Note that this chapter requires some familiarity with orders and topology. The necessary concepts are introduced in Appendices B.1 and B.2.

A preliminary version of this work appeared in [Sta00b] and [Sta01].

Overview. The chapter starts with a brief explanation of an abstract model of the electronic height control system (EHC) which is used as an example in the sequel (Section 6.1). Section 6.2 defines the underlying system model which is the basis for the rest of this chapter. Section 6.3 lists and defines the properties which resulted from our evaluation of case studies and textbooks. Furthermore, it contains the classification of the properties, sorts them according to their relevance as derived from the case studies and examines the utility of trace inclusion as refinement notion for hybrid systems. In Section 6.4, some proof concepts for stability and attraction are introduced and parallels to computer science are drawn. The proof methods are applied to two examples in Section 6.5. Section 6.6 discusses this chapter's contribution, compares it with related work and outlines future work.

6.1 Example

Throughout this chapter we will use the following model of the EHC system (Section 1.4) as a motivational example. As the properties we will outline can (and should) already be considered early in the development, we consider a model with only one component here, which contains a very abstract model of the control logic and the physics of the suspension system. Figure 6.1 depicts the component's interface. Channel *height* outputs the chassis level, input *dist* denotes external disturbances and channel *bend* reports whether a curve is entered or left. Channels *height* and *dist* are continuous channels, and *bend* is a discrete channel which only transmits events. Figure 6.2 defines the component's behavior as a HySChart. The definitions of actions, invariants and activities is given in Section 6.5.1, where two properties of the system are regarded in detail.

Figure 6.1: HyAChart of the EHC system.

Informally, the behavior of the system is as follows. Whenever an event *bend*, which signals entry/exit of a bend, is received control changes from *outBend* to *inBend* and vice versa. In *inBend* the system's actuators, a compressor and an escape valve, are off and the chassis level is not influenced. (This also holds in the degenerate case where a bend is never left.) Hierarchic state *outBend* has three substates. If the chassis level

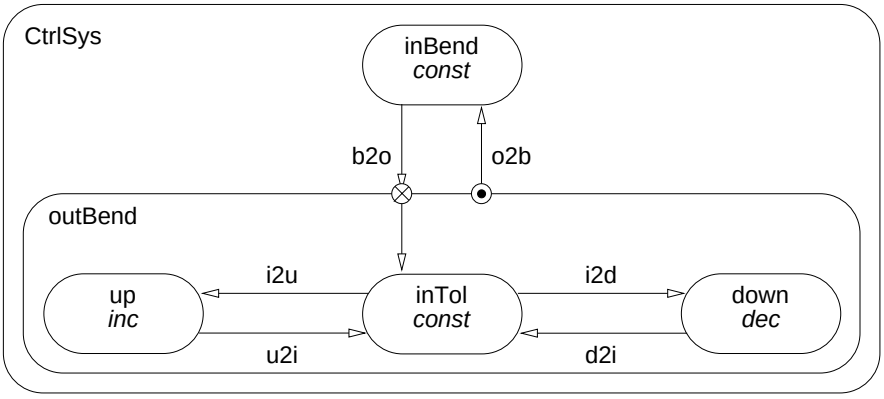


Figure 6.2: HySChart for the control system *CtrlSys*.

height is in the desired *tolerance interval*, control is in *inTol* and the actuators are off. If it is too high, control is in *down* and the escape valve is open thereby influencing *height*. Otherwise, if *height* is too low, control is in *up* and the compressor is on.

6.2 Systems under Consideration

The properties presented in this chapter all assume a system structure like the one depicted in Figure 6.3. Its basic elements are a controlling device (Controller), the

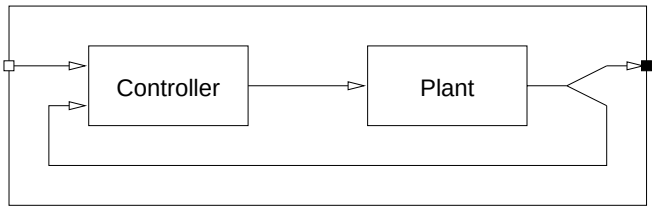


Figure 6.3: General structure of a closed-loop control system.

physical environment (Plant) and a feedback loop. Such systems are called feedback control systems. If there is no feedback from the plant's output to the controlling device, the system is called a *supervisory* or *open loop* control system [PH88].

On a very abstract level we can regard a system as a nondeterministic function mapping a *cause* to a (nonempty) set of possible *effects*:[1]

$$Sys \in \mathcal{C} \to \mathcal{P}(\mathcal{E})$$

[1]Although control theory usually focuses on deterministic systems, we employ a more general nondeterministic approach which is closer to models in the field of computer science.

where $\mathcal{C}$ denotes the set of causes and $\mathcal{E}$ denotes the set of effects. Usually, causes are some sort of input to the system and effects are the produced output (see below). Pairs (c, e) with $e \in Sys(c)$ are called *behaviors* of the system.

Black box behavior. In the following we will use two specializations of this system model. The first one describes the black box behavior (or I/O behavior) of a system as a function:

$$Sys_{IO} \in \mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_+})$$

where the notational convention is as in the preceding chapters: $\mathcal{I}$ is the input domain and $\mathcal{O}$ is the output domain, i.e. $\mathcal{C}$ has been instantiated to $\mathcal{I}^{\mathbb{R}_+}$ and $\mathcal{E}$ to $\mathcal{O}^{\mathbb{R}_+}$, where $\alpha^{\mathbb{R}_+}$ denotes the set of functions from the non-negative real numbers $\mathbb{R}_+$ to α.[2] Like before, elements of $\alpha^{\mathbb{R}_+}$ are called *streams*, *traces* or *trajectories* and functions on streams, like Sys_{IO}, are also called *stream processing functions* [Bro97b]. We require Sys_{IO} to be total in its input, i.e. $Sys_{IO}(\iota) \neq \emptyset$ for all $\iota \in \mathcal{I}^{\mathbb{R}_+}$. Furthermore, to model realistic systems Sys_{IO} must be *(weakly) time guarded*, i.e. its output up to any time t may not depend on future input:

$$\iota_1|_{[0,t)} = \iota_2|_{[0,t)} \quad \Rightarrow \quad Sys_{IO}(\iota_1)|_{[0,t)} = Sys_{IO}(\iota_2)|_{[0,t)}$$

where $\xi|_M$ denotes the restriction of function ξ to arguments in the set M and is extended to sets of functions in a pointwise manner. Throughout the chapter variable t is used to denote a point in time, $t \in \mathbb{R}_+$.

In the example of Section 6.1 the input domain $\mathcal{I}$ is $\mathbb{R}_+ \times \mathbb{R}$ for the inputs *bend* and *dist*, and the output domain $\mathcal{O}$ is $\mathbb{R}$ for the output *height*.

White box behavior. Some of the properties we will examine require a state-based (or white box) system description, or come in two variants, one depending only on the interface behavior and the other depending on the system state. We formalize state-based systems as follows:

$$Sys_S \in \mathcal{S} \to (\mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{S}^{\mathbb{R}_+}))$$
$$Out \in \mathcal{S} \to \mathcal{O}$$

Again $\mathcal{I}$ is the input domain, $\mathcal{S}$ is the state space. Depending on the initial state in $\mathcal{S}$, Sys_S maps an input trajectory to a set of possible state trajectories. Thus, $\mathcal{C}$ is instantiated to $\mathcal{S} \times \mathcal{I}^{\mathbb{R}_+}$ and $\mathcal{E}$ to $\mathcal{S}^{\mathbb{R}_+}$, here. All state trajectories are required to start with the prescribed initial state, $\forall \sigma \in Sys_S(s, \iota).\, \sigma(0) = s$. Function Out maps the current state to the current output in $\mathcal{O}$. Thus, it performs some state-to-output conversion. It is extended in time by pointwise extension, $Out^\dagger(\sigma)(t) = Out(\sigma(t))$ for all $t \in \mathbb{R}_+$. In the case of HyCharts, Out simply is a projection which hides the private variables (Section 3.2.1). The black box behavior of a state-based system with initial state s is defined by the sequential composition of $Sys_S(s)$ and the time extension of Out:

[2] The continuity restrictions enforced in preceding chapters are not required here.

$$Sys_{IO} = Sys_S(s); Out^\dagger$$

where sequential composition of a nondeterministic system A and a deterministic system B is defined as $(A;B)(\alpha) = \{\gamma \mid \exists \beta.\, \beta \in A(\alpha) \ \wedge \ \gamma = B(\beta)\}$. For a set of initial states $S \subseteq \mathcal{S}$ we define $Sys_S(S)$ by pointwise extension. As with black box descriptions, we require that state-based system descriptions $Sys_S(s)$ be total in the input and time guarded for any start state s.

Furthermore, we demand that Sys_S is *time invariant*, meaning that the system does not depend on absolute timing. Instead, a system's state trajectories are completely determined by the initial state and the input trajectory. Formally, we demand that a left shift of the input results in the same left shift of the state trajectory for the shifted start state:

$$\sigma \in Sys_S(s, \iota) \Rightarrow \sigma^{-u}|_{\mathbb{R}_+} \in Sys_S(\sigma(u), \iota^{-u}|_{\mathbb{R}_+})$$

for any $u \geq 0$, where φ^{-u} is the left shift of φ by u, $\varphi^{-u}(t) = \varphi(u+t)$. Because of time invariance, it is sensible to regard a disturbance of a system, i.e. an unforeseen change in its state, as reinitialization of the system. Hence, the behavior of system Sys_S resulting from disturbing it by setting its state to s' at time t is defined by $Sys_{disturbed}(s_0, \iota, s', t) = \{\sigma \mid \exists \sigma_1 \in Sys_S(s_0, \iota).\, \exists \sigma_2 \in Sys_S(s', \iota^{-t}).\, \sigma|_{[0,t)} = \sigma_1|_{[0,t)} \ \wedge \ \sigma^{-t}|_{\mathbb{R}_+} = \sigma_2\}$. When we consider a system's reaction to disturbances, it therefore suffices to consider disturbances of the initial state. System behavior after later disturbances can be inferred from its reaction to initial disturbances.

Note that systems with a semantics of one of the above two types, black box or white box, can be specified with HyCharts. In the HyChart context we denoted the black box behavior of a component by *Cmp* and its white box behavior by *St* (Section 3.2.4).

6.3 Properties

Based on the system model introduced above, a number of important properties of control systems are defined and classified in this section. Relative to this classification we then discuss the utility of trace inclusion as refinement notion. The properties have been extracted from the evaluation of nine hybrid system case studies, which were taken from papers and from student projects performed within various departments of the Technische Universität München [BHKT98, Eng97, Ger97, Sta97, Rap98, Ant96, Abr96, SB98, BGM93], and from textbooks on control theory, in particular [Föl90, Föl87, PH88, Vac95, Lei87]. Case studies were used as one source in order to be able to estimate the practical relevance of the properties.

The case studies. The topics and domains of the case studies are as follows. [BHKT98] introduces a hybrid benchmark problem from the field of chemical engineering and lists required properties of the proposed system. The introductory paper [Eng97] explains hybrid systems along an example from process automation and lists

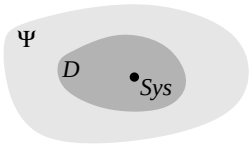


Figure 6.4: Robustness.

system properties whose analysis is required to be feasible with present and future methods for hybrid systems. In [Ger97], elements of the design of an autonomous cruise control system, which adjusts a car's speed and distance to the car in front, are discussed and vital properties are analyzed analytically and with prototypes, respectively. [Sta97] employs the *HyTech* model checker [HHWT95] to verify some properties of the EHC system which is also used as running example throughout this thesis. The version we regard is an abstraction of that in [Sta97]. In [Rap98], a further system stemming from automotive electronics is modeled with MatrixX/Betterstate [Int00, Int98]. [Ant96] models a system of conveyor belts from the process automation domain. [Abr96] introduces a steam boiler example as a benchmark for formal methods for embedded systems. In [SB98], a problem from robotics is discussed and an architecture for its solution is developed and validated with simulations. [BGM93] analyzes the behavior of a control system for a hopping robot by systematic simulation.

Formalization. Unless otherwise mentioned, all the following definitions assume that systems are given as nondeterministic functions from a cause to a set of effects. Thus, they apply to I/O-based as well as to state-based system descriptions.

6.3.1 Robustness

An important property of control systems is that they satisfy some required properties in spite of deviations between the model of the system and the real system [Fri86]. Deviations may range from inaccurate parameters, caused, for instance, by aging effects, to structural mistakes, such as an incorrect model of the plant.

Hence, robustness of a system depends on the system itself, the regarded property, and the regarded deviations. Let D be the set of systems deviating from system Sys in the way that is supposed to be considered. For instance, D may be defined relying upon some metric on systems. We demand that $Sys \in D$, i.e. no deviation from the perfect system is also allowed. Furthermore, let $valid_\Psi(S)$ be an evaluation function which is true iff system S satisfies property Ψ. We define robustness as below.

Definition 6.1 (Robustness.) *A system Sys with deviations D, $Sys \in D$,* robustly satisfies *property Ψ iff $\forall Sys' \in D.\, valid_\Psi(Sys')$ holds.*

Thus, robustness is parameterized with the notion of validity $valid_\Psi(S)$ of the considered object property Ψ. Robustness is visualized in Figure 6.4.

[MT75] introduces the related notion of structural stability. There, a topology is defined on the set of deviating systems. Based on this topology, the authors demand that small deviations in the system result in small output deviations (see also Section 6.3.3). A further parallel can be seen to numerical mathematics. When we interpret Ψ as the specification of a problem, the system as an algorithm which computes an exact solution to the problem and the set of deviating systems as the same algorithm afflicted with roundoff errors, robustness is related to the notion of stability of algorithms used in numerical mathematics [BSMM97]. This notion expresses that the result of the algorithm with roundoff errors is a solution to the problem for slightly modified input.

Note that robustness is related to nondeterminism. For universal properties, i.e. properties which have to hold for any execution of a system, we may subsume the behavior of all the deviating systems in one nondeterministic system, $Sys'(c) = \bigcup_{Sys \in D} Sys(c)$ for causes $c \in \mathcal{C}$. To prove robustness of the universal property w.r.t. D, it then suffices to show that Sys' satisfies the property.

As a typical, simple example for robustness, think of a system Sys given by a differential equation. Allowed tolerances associated with the constants in the equation lead to set D of deviating systems. The property Ψ to be satisfied robustly could express that some quantity always evolves within given bounds. In the context of our EHC system, an example for robustness is to demand that stability of the system (see Section 6.3.3) is also satisfied for a range of system variants containing actuators with differing performance characteristics.

6.3.2 Optimality

Apart from finding a solution to a given problem, control theory is interested in identifying the optimal solution for the problem w.r.t. some cost function. Among others, possible aims are minimizing energy consumption or maximizing throughput of a plant.

For a given system Sys, a set of alternative systems A and a cost function cf from the set of systems to a linearly ordered set with order $<$, we define optimality as follows.

Definition 6.2 (Optimality.) *System Sys is* optimal *w.r.t. alternatives A and cost function cf iff* $\forall Sys' \in A.\ cf(Sys) \leq cf(Sys')$.

Here, $\leq$ denotes the reflexive closure of $<$. For instance, the set of alternative systems A may be defined as the set of all those systems which satisfy a given abstract specification. In practice, cost functions are often based on the system's output or state. For the EHC system, an optimality property would be to require that the compressor is operated such that its energy consumption is minimal. Note that this would necessitate to include the system's energy consumption into our model of the EHC.

6.3.3 Stability

The general idea of stability is based on the desire that small disturbances in the causes should only result in small disturbances in the effects. To formalize closeness of one cause (or effect) to another, we use the notion of neighborhood which is induced by the topologies considered for causes and effects (see Appendix B.2 for basic concepts of topology). We write $N(\alpha)$ for the set of all neighborhoods of α and $\mathfrak{O}_X$ for the topology on X.

Definition 6.3 ((General) Stability.) *For a system* $Sys \in \mathcal{C} \to \mathcal{P}(\mathcal{E})$ *between topological spaces* $(\mathcal{C}, \mathfrak{O}_\mathcal{C})$ *and* $(\mathcal{E}, \mathfrak{O}_\mathcal{E})$*, the tuple of sets of causes and effects* (C, E) *with* $C \subseteq \mathcal{C}$ *and* $E \subseteq \mathcal{E}$ *is* stable *w.r.t. these spaces iff* $\forall \alpha \in N(E). \exists \beta \in N(C). \forall b \in \beta. Sys(b) \subseteq \alpha$.

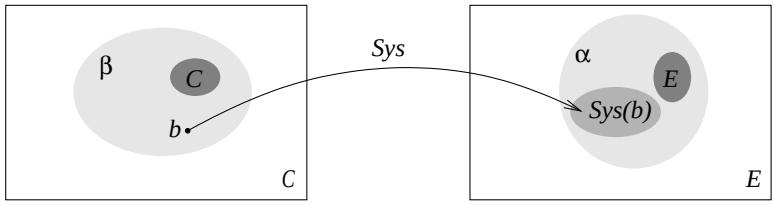


Figure 6.5: General stability.

Figure 6.5 visualizes the situation. The definition requires that for any neighborhood of the effects E there is a neighborhood of the causes C such that the effects resulting from any of these causes are in the considered neighborhood of E. Note that disturbances and their "size" are a vague concept. The stability definition tries to grasp the notion of "small disturbances of causes" resulting in "small disturbances of effects" by universally quantifying over the neighborhoods $N(E)$. For a "small" neighborhood $\alpha \in N(E)$, i.e. one that does not contain much more elements than E, stability provides that there is a $\beta \in N(C)$, which may also be small, such that disturbed causes in β result in disturbances of the effects which are in α. Hence, stability allows the conclusion that there is a neighborhood of C in which disturbances of causes must be contained in order to ensure that the resulting disturbances of the effects remain within a desired neighborhood of E.

Mesarovic et al. [MT75] furthermore demand that $Sys(C) \subseteq E$ which, as we will see in Section 6.3.3.2, corresponds to the stability of invariant sets in the state-based case. If enforced, this demand excludes that (C, E) is stable although Sys maps C to a set which is disjoint from E but contained in every neighborhood of E.

In topologies where C and E are open, (C, E) is stable iff $\forall c \in C. Sys(c) \subseteq E$ (for the proof see Theorem A.13 in Appendix A.3). In particular, this includes the discrete topology (cf. Appendix B.2). This result is a consequence of the definition of neighborhood which for open sets X permits $X \in N(X)$ (see Definition B.6).

	$C \subseteq .$	$E \subseteq .$				
$\mathrm{Tr}_{\mathrm{IO}}$	$\mathcal{I}^{\mathbb{R}_+}$	$\mathcal{O}^{\mathbb{R}_+}$	$\forall \alpha \in N(E).\, \exists \beta \in N(C).$		$\forall c \in \beta.$	$Sys_{IO}(c) \subseteq \alpha$
Tr_{S}	$\mathcal{S}$	$\mathcal{S}^{\mathbb{R}_+}$		$\forall \iota \in I.$		$Sys_{S}(c, \iota) \subseteq \alpha$
$\mathrm{Pt}_{\mathrm{IO}}$	$\mathcal{I}^{\mathbb{R}_+}$	$\mathcal{O}$				$Sys_{IO}(c) \subseteq \alpha^{\mathbb{R}_+}$
Pt_{S}	$C = E \subseteq \mathcal{S}$			$\forall \iota \in I.$		$Sys_{S}(c, \iota) \subseteq \alpha^{\mathbb{R}_+}$

Table 6.1: Notions of stability.

However, note that [MT75] do not permit $X \in N(X)$ in their definition of stability. In the standard case of control theory where stability of points is studied w.r.t. the usual topology on the Euclidean space, such singletons are closed sets and hence the definition of neighborhoods for open sets is immaterial there.

We remark that in this general form the concept of stability is related to the continuity of functions on the reals. In standard textbooks on Analysis, continuity of a function on the reals $f \in \mathbb{R} \to \mathbb{R}$ at x_0 is usually defined as $\forall \varepsilon > 0.\, \exists \delta > 0.\, \forall x.\, |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \varepsilon$ [Kön90]. In terms of neighborhoods on the reals, this can be expressed as $\forall \alpha \in N(f(x_0)).\, \exists \beta \in N(x_0).\, \forall x.\, x \in \beta \Rightarrow f(x) \in \alpha$ which corresponds to the general notion of stability from above.

In the following sections we will outline a number of specializations of this general stability definition.

6.3.3.1 Stability of Trajectories

Depending on the instantiation of causes and effects in the general definition of stability from above and depending on employing an I/O-based or a state-based system model, we obtain a number of different notions of stability.

Stability of (sets of) trajectories basically expresses that the *traces* of the system are in the neighborhood of some desired traces for small disturbances in the system's causes. An I/O-based and a state-based formalization for stability of trajectories of a tuple (C, E) of sets of causes and effects is given in the first two lines of Table 6.1, labeled $\mathrm{Tr}_{\mathrm{IO}}$ and Tr_{S}, respectively. Note that the considered effects are traces here. Furthermore, note that for state-based system descriptions, our definition is parameterized with the set of inputs $I \subseteq \mathcal{I}^{\mathbb{R}_+}$ for which stability must hold. This differs from usual definitions like those in [MT75, MW95], which completely disregard external input, for state-based system descriptions. In practice, I may be used to restrict the input to continuous trajectories or to require that inputs are constantly zero (see Section 6.5.1).

Stability of trajectories is of particular interest in connection with so called *limit cycles*. We will consider this in see Section 6.3.5.2.

Poisson stability. A rather different notion of stability is referred to as *Poisson stability* [Lei87]. Informally, a system is stable in the sense of Poisson iff there is a point

in the system's trajectory whose neighborhoods are visited by the trajectory infinitely often. Poisson stability is related to *reactivity* in computer science (see below).

For I/O-based systems Poisson stability can be defined as follows. (A definition for state-based systems is similar.)

Definition 6.4 (Poisson stability.) *For a system* $Sys_{IO} \in \mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_+})$ *on the topological space* $(\mathcal{O}, \mathfrak{O}_{\mathcal{O}})$ *a set of trajectories* $Sys_{IO}(\iota)$ *is* positively Poisson stable *iff* $\exists t_0. \forall \alpha \in N(Sys_{IO}(\iota)(t_0)). \forall t. \exists t' \geq t. Sys_{IO}(\iota)(t') \subseteq \alpha$.

According to Leipholz, an application area for Poisson stability is the stability of planet orbits in astronomy [Lei87]. There it is usually required that *every* point of the orbit is visited infinitely often.

In topologies where the $Sys_{IO}(\iota)(t)$ are open sets for all $t \in \mathbb{R}_+$, Poisson stability has some similarity to *reactivity properties* in computer science, as defined in [CMP91]. Informally, reactivity properties express that a certain predicate *always* holds *eventually*, i.e. it must hold infinitely often. In topologies where the $Sys_{IO}(\iota)(t)$ are open, Poisson stability similarly expresses that some set $Sys_{IO}(\iota)(t)$ is visited infinitely often. Hence, Poisson stability may be regarded as expressing a reactivity property. However, reactivity properties in general are not restricted to expressing that some already visited set of states is visited infinitely often. Instead the set of states which is required to be visited infinitely often can be defined independently from the system's trajectories.

A second parallel to computer science is to Büchi automata [Tho90]. A Büchi automaton can be used to define infinite sequences of symbols which contain certain finite subsequences infinitely often.

6.3.3.2 Stability of Points

Like stability of trajectories, stability of (sets of) points can also be derived from Definition 6.3. For stability of points, however, a notion of neighborhood of points instead of neighborhood of trajectories is used.

Informally, stability of (sets of) points expresses that the effects of a system *always* are in the neighborhood of some desired points for small disturbances in the system's causes. In contrast to stability of trajectories which regards traces as a whole, stability of points views the effects of a system in a pointwise manner, namely for each point in time. In the sense of Leipholz [Lei87] stability of points is a purely *geometric* notion of stability, while stability of trajectories is a *kinematic* notion of stability, because is also consideres the time at which the individual points are reached. Depending on the topology for trajectories stability of (sets of) points and stability of trajectories are equivalent, namely in topologies which only consider the points visited and not the time when they are visited. The formalization of stability of points of a tuple (C, E)

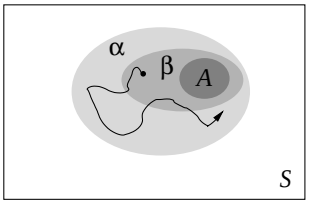


Figure 6.6: State-based stability of points.

of sets of causes and effects is given in the bottom two lines of Table 6.1 for I/O-based (label $\mathrm{Pt_{IO}}$) and state-based systems (label $\mathrm{Pt_S}$), respectively.[3] Note that the considered effects are points here, and the system's output is regarded in a pointwise manner.

State-based stability of points. In the following we will focus on state-based stability of (sets of) points, which is the stability notion encountered most frequently in applications. It is concerned with the effect of disturbances in the system state. According to the Table 6.1, for a state-based system $Sys_S \in \mathcal{S} \to (\mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{S}^{\mathbb{R}_+}))$ a set $A \subseteq \mathcal{S}$ is stable w.r.t. the topology $\mathfrak{O}_\mathcal{S}$ on $\mathcal{S}$ and the inputs $I \subseteq \mathcal{I}^{\mathbb{R}_+}$ iff the formula in the fourth line of the table holds (for $A = C = E$). Again note that the definition is parameterized with the set of inputs I for which stability must hold. Figure 6.6 visualizes this notion of stability along an example trajectory: For neighborhood α of A there is neighborhood β such that the trajectory starting in β never leaves α.

While the definition only consideres disturbances in a system's initial state explicitly, we can infer that disturbances in the state at any point in time cause the same (shifted) behavior as a likewise disturbed initial state. This is due to our definition of disturbed systems which in turn is motivated by time invariance of the considered systems (Section 6.2).

Liapunov stability as defined e.g. in [Lei87] is a specialization of our definition to deterministic systems and the natural metric on the real line. Typically Liapunov stability of a point $x \in \mathbb{R}$ is defined as follows: $\forall \varepsilon > 0.\, \exists \delta > 0.\, \forall x_0.\, |x - x_0| < \delta \Rightarrow \forall t > 0.\, |x - Sys_S(x_0)(t)| < \varepsilon$, where Sys_S is deterministic and gets no external input. If we take $\{x\}$ as the considered set of causes and effects and if we employ the notion of neighborhood that is induced by taking, as usually, the sets $B_\delta(y) = \{y' \mid |y - y'| < \delta\}$ as base for the topology on $\mathbb{R}$, the equivalence to our definition of stability is straightforward.

The stability definitions in [MT75, MW95] additionally require that the regarded set is invariant, i.e. that it is never left again once it has been entered.

[3] As α is a set of points in the bottom two lines of the table, $\alpha^{\mathbb{R}_+}$ as usual denotes the set of functions from $\mathbb{R}_+$ to α.

Definition 6.5 (Invariance.) *A set* $A \subseteq \mathcal{S}$ *is* invariant *for system* Sys_S *and the inputs* $I \subseteq \mathcal{I}^{\mathbb{R}+}$ *iff* $\forall s \in A. \forall \iota \in I.\, Sys_S(s, \iota) \subseteq A^{\mathbb{R}+}$.

In topologies where A is open, invariance of A is equivalent to the stability of A (see Theorem A.14 in Appendix A.3). When associating disturbances with neighborhoods this can informally be explained as follows. In topologies of this kind there is a neighborhood of A (i.e. a disturbance of A) which does not contain elements outside A. Hence, the smallest possible disturbance is having no disturbance at all. Or, more informally, there are no "small" disturbances which leave A. This suits well to discrete systems where all disturbances can be regarded as equally big, i.e. for discrete systems the only small disturbance also is having no disturbance at all. From a computer science perspective invariance of A is a safety property [CMP91]. Hence, in topologies where A is open stability of A is a safety property.

A notion contrary to stability is *chaos* [Wig90]. One precondition for chaotic behavior is sensitive dependence on initial conditions for an invariant set A. Informally, this expresses that for any two arbitrarily close points in A system traces starting in them separate after some time.

As an example we will prove state-based stability of a set A for the EHC system in Section 6.5.1. The considered set contains the tolerance interval, in which the chassis level is desired to be, and stability is regarded w.r.t. a set of specific inputs I. The topology used there is similar to the natural topology on $\mathbb{R}$, but coarser. It is motivated by the application.

6.3.4 Attraction

Often we do not only want that disturbances only have a limited effect (stability), but also that a system returns to some desired trajectory or state after a disturbance. Informally, attraction of a set means that despite of some initial disturbance the set is always reached after a finite or infinite amount of time. Like for stability, a number of variants of attraction result depending on whether we regard attraction of trajectories or attraction of points for I/O-based or for state-based systems. Table 6.3 contains the formulas defining these variants of attraction and Table 6.2 lists the instantiation of causes and effects for each line of Table 6.3, and the resulting notion of attraction. In the following we give an overview over these notions and will then consider attraction of points in the state-based case, which occurs frequently, in more detail. As we will see in Section 6.5.2, attraction is related to a property called *self-stabilization* in computer science.

6.3.4.1 Attraction of Trajectories

In practice, attraction of trajectories is often of particular interest for periodic trajectories, expressing e.g. the repeated execution of some required task. For I/O-based

	$C \subseteq .$	$E \subseteq .$	stability/attraction of (sets of) ...
1	$\mathcal{I}^{\mathbb{R}_+}$	$\mathcal{O}^{\mathbb{R}_+}$	trajectories, I/O-based
2	$\mathcal{S}$	$\mathcal{S}^{\mathbb{R}_+}$	trajectories, state-based
3	$\mathcal{I}^{\mathbb{R}_+}$	$\mathcal{O}$	points, I/O-based
4	$C = E \subseteq \mathcal{S}$		points, state-based

Table 6.2: Instantiations of causes and effects and resulting notion of stability and attraction.

1	$\exists \alpha \in N(C).$		$\forall c \in \alpha. \forall \beta \in N(E).$	$\forall e \in Sys_{IO}(c).$	$\exists t.$	$e\|_{[t,\infty)} \in \beta\|_{[t,\infty)}$
2		$\forall \iota \in I.$		$\forall e \in Sys_S(c, \iota).$		
3				$\forall e \in Sys_{IO}(c).$		$e\|_{[t,\infty)} \in \beta^{\mathbb{R}_+}\|_{[t,\infty)}$
4		$\forall \iota \in I.$		$\forall e \in Sys_S(c, \iota).$		

Table 6.3: Notions of attraction.

systems attraction of trajectories is defined as follows.

Definition 6.6 (Attractive trajectories (I/O-based).) *For an I/O-based system Sys_{IO} the set of output trajectories $E \subseteq \mathcal{O}^{\mathbb{R}_+}$ is attractive w.r.t. the set of input trajectories $C \subseteq \mathcal{I}^{\mathbb{R}_+}$ and the topologies on $\mathcal{O}^{\mathbb{R}_+}$ and $\mathcal{I}^{\mathbb{R}_+}$ iff $\exists \alpha \in N(C). \forall c \in \alpha. \forall \beta \in N(E). \forall e \in Sys_{IO}(c). \exists t. e|_{[t,\infty)} \in \beta|_{[t,\infty)}$ (first line of Table 6.3).*

This requires that there is a neighborhood α of the input trajectories C such that for every input in α and for every neighborhood β of the output trajectories E the system's reaction to this inputs is such that for each of the system's output trajectories there is a point in time t such that the output trajectory is in β for the remaining time $[t, \infty)$. From a more abstract point of view the formula expresses that for some neighborhood of the input trajectories the system's output converges to the dynamics described by E as time progresses.

The state-based version of attraction of sets of trajectories is similar. Here, however, the set of causes consists of initial states, the set of effects consists of traces in the state space and the definition is parameterized with the set of inputs I for with attraction is supposed to hold (see line 2 of Table 6.3 for the formula).

6.3.4.2 Attraction of Points

Attraction of trajectories requires that the evolution of the effects converges to the attractive trajectories. If the attractive trajectory e.g. is a sine function, this means that the system's output must become similar to this sine as time progresses.[4] In contrast attraction of (a set of) points denotes that the effects must reach every neighborhood of the considered points as time progresses. It does not constrain the evolution of the output within this set. In the bottom two lines of the formalizations

[4]The notion of similarity results from the underlying topology on the set of causes.

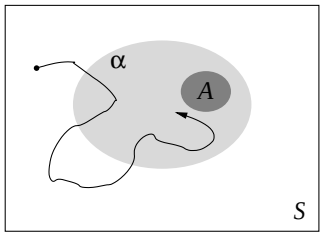


Figure 6.7: Global attraction of A.

in Table 6.3 this becomes apparent in the last column. Here, it is only required that the rest of the output is some trajectory that completely lies within neighborhood β of the set of effects E, which is denoted by considering all trajectories over β, $\beta^{\mathbb{R}_+}$, in the formula. Hence, like for stability, attraction of points can be regarded as geometric notion of attraction whereas attraction of trajectories may be seen as a kinematic notion.

We do not go into detail on the I/O-based version of attraction of (sets of) points which is formalized in the third line of Table 6.3, but focus on the state-based version now, because it is encountered most frequently in control theory.

State-based attraction of points. For state-based systems attraction of points is defined as follows.

Definition 6.7 (Attractive set.) *The set $A \subseteq \mathcal{S}$ is* attractive *w.r.t. the topology $\mathfrak{O}_\mathcal{S}$ on $\mathcal{S}$ and the inputs $I \subseteq \mathcal{I}^{\mathbb{R}_+}$ iff $\exists \alpha \in N(A).\, \forall \iota \in I.\, \forall s \in \alpha.\, \forall \beta \in N(A).\, \forall \sigma \in Sys_S(s,\iota).\, \exists t.\, \forall t' \geq t.\, \sigma(t') \in \beta$.*

A is globally attractive *iff $\forall \iota \in I.\, \forall s \in \mathcal{S}.\, \forall \alpha \in N(A).\, \forall \sigma \in Sys_S(s,\iota).\, \exists t.\, \forall t' \geq t.\, \sigma(t') \in \alpha$.*

Global attraction means that each system behavior remains inside any neighborhood of A eventually for *any* starting state. Hence, it expresses a kind of convergence to set A. (Normal) attraction only requires that there is a neighborhood of A such that system behaviors starting in it exhibit this convergence to A. Again we have parameterized our definition of attraction with a set I of allowed external inputs. Figure 6.7 visualizes global attraction. Note that the depicted trajectory leaves α before reentering it and not leaving it again thereafter. This does not violate attraction.

In topologies where A is open global attraction is equivalent to the property that A is already reached in finite time ($t \in \mathbb{R}_+$) and not left again, i.e. to $\forall \iota \in I.\, \forall s \in \mathcal{S}.\, \forall \sigma \in Sys_S(s,\iota).\, \exists t.\, \forall t' \geq t.\, \sigma(t') \in A$. Hence, the notion of asymptotically approaching A expressed via the neighborhoods in the definition of attraction is replaced by truly reaching A eventually in this property (Theorem A.15 in Appendix A.3). Again, this suits well to discrete systems. The idea of attraction is that we are ensured to get arbitrarily close to the attractive set A. For discrete systems all states different from A

can be regarded as being far away from it. The only states close to A are those inside it. Thus, for topologies where A is open attraction is a *persistence property* [CMP91] when regarded from a computer science point of view, i.e. it expresses that "*eventually* the system *always* remains in A". Persistence properties contain a safety as well as a liveness part. In the example of Section 6.5.2 we will encounter a discrete system with a persistence property (self-stabilization) which can be expressed as attraction of a set A on a topology where A is open.

Definition 6.8 (Asymptotic stability.) *A is called* asymptotically stable *iff it is stable and attractive. A is called* asymptotically stable in the large *iff it is stable and globally attractive.*

Note that in our general setting attraction does not imply stability. The reason is that attraction is interested in system behavior as time goes to infinity, whereas stability considers the whole time axis. An example can be constructed along Figure 6.7. If the topology on $\mathcal{S}$ in the figure is such that α is the only neighborhood of A and we consider the depicted trajectory without the initial segment outside α, then this trajectory still satisfies attraction. Nevertheless, stability is violated, because there is no neighborhood of A (α is the only one) such that all trajectories starting in it never leave α. However, for most control systems studied in practice attraction implies stability. Therefore, many control theory textbooks do not introduce the notion of attraction for its own, but only introduce asymptotic stability.

Asymptotic stability in the sense of Liapunov is a specialization of our definition to deterministic systems and the natural metric on the real line. For a stable point x asymptotic stability usually requires that the following holds [Lue79]: $\exists \delta > 0. \forall x_0. |x - x_0| < \delta \Rightarrow \lim_{t\to\infty} Sys_S(x_0)(t) = x$, where Sys_S is deterministic and gets no external input. To see that this is a special case of the definition in line four of Table 6.3, we need to take $\{x\}$ as the considered set of causes and effects, and employ the natural topology on the real line as outlined in Section 6.3.3.2. As $\lim_{t\to\infty} Sys_S(x_0)(t) = x$ is equivalent to $\forall \varepsilon > 0. \exists t'. \forall t \geq t'. |Sys_S(x_0)(t) - x| < \varepsilon$, the correspondence to our definition is straightforward.

In Section 6.5.1 we prove state-based attraction of a set A containing the EHC system's tolerance interval. The claim is limited to a specific set of inputs I, and A is a closed set w.r.t. the topology regarded in the example.

6.3.5 Further Properties

Whereas the properties above are rather general and meaningful for most systems, there are a lot of further, more detailed requirements on individual systems. Classes of such properties will be discussed in the following.

6.3.5.1 Universal Properties

A characteristic of many properties is that they constrain *all* possible behaviors of a system. Formally, such properties can be written as $\forall c \in \mathcal{C}.\forall e \in Sys(c).(c,e) \in \Phi$, where $\Phi \subseteq \mathcal{C} \times \mathcal{E}$. In computer science properties of this kind are often formalized with linear time temporal logics (LTL) [MP92].

A very important example for such properties are invariants which demand that a certain time-independent constraint be always satisfied [CMP91]. Invariance properties correspond to the invariance of sets as defined in Definition 6.5. Typically invariants constrain the range of state variables, e.g. an invariant could demand that the temperature in a nuclear reactor never exceeds a certain threshold. An invariant of the EHC system would be that the actuators, compressor and escape valve, are not used simultaneously. Further examples for properties expressable in LTL are bounded response requirements, and safety and liveness formulas in general [CMP91]. For the EHC system a liveness property e.g. is that state *inBend* is entered some time after a *bend* event.

6.3.5.2 Existential Properties

Existential properties can be divided into two relevant groups. First, we have properties which demand the existence of certain behavior. Such properties involve existential quantification over causes and effects (see (1) below). Second, there are properties which require that causes exist which are guaranteed to lead to some desired effect. For these properties existential quantification over the causes and universal quantification over the effects is involved (see (2) below).

(1) Existence of behavior. The only property of this kind which was regarded in some of the evaluated case studies is the existence of periodic behavior when there are no external disturbances. For instance, this property is relevant for the hopping robot [BGM93], which is required to hop constantly as long as it is not perturbed from outside.

For state-based system descriptions the existence of periodic behavior can formally be written as follows

$$\exists s_0.\exists \sigma \in Sys_S(s_0,0).\,\exists t.\,\exists \tau \in \mathcal{O}^{[0,t)}.\,\sigma = \tau^\infty$$

where $0 \in \mathcal{I}^{\mathbb{R}+}$ denotes a neutral input, i.e. no external disturbances, $\mathcal{O}^{[0,t)} = [0,t) \to \mathcal{O}$, and τ^∞ denotes the trajectory resulting from the infinite repetition of τ. An I/O-based version of this property can be given in a similar manner. Such periodic trajectories in reaction to neutral input are called *limit cycles* [Lei87]. In practice, the non-existence of stable limit cycles is sometimes used as an indirect evidence for the stability of the invariant sets of non-linear control systems. [Föl87] states that,

as a rule of thumb, the invariant sets of non-linear control systems are asymptotically stable in the large if no stable limit cycles exists.

In an analogy to computer science the existential path quantifier of CTL [Eme90] can be used to express similar existential properties.

(2) Existence of Causes. In the regarded case studies no property of this type was examined. However, the properties of *controllability* and *observability*, which are important for state based controller design [Oga87, Föl90, Son90, PH88], are of this kind. In contrast to all the properties regarded before, they only refer to the plant, not to the whole control system, and they presuppose a state based model of the plant. Thus, the term system refers to the plant in the remainder of this paragraph.

Controllability means that there is an input trajectory such that a certain state can be reached from the initial state within finite time. In an adaption from [Son90] to nondeterministic systems, we can define controllability from state set S_1 to state set S_2 as follows:

$$\exists t.\, \exists \iota \in \mathcal{I}^{\mathbb{R}_+}.\, \forall s \in S_1.\, Sys_S(s,\iota)(t) \subseteq S_2$$

The input trajectory after time t is irrelevant for controllability, since we are dealing with time guarded systems. If S_2 is a one element set, this is similar to the definition of controllability for deterministic systems [Son90], because we required that Sys be total in its inputs. A system is called *fully controllable* iff any system state can be controlled to any other system state.

Observability denotes that for any two distinct states there is an input such that a finite observation of the system output suffices to detect the distinctness of the states. Like in our treatment of controllability we define observability (or *distinguishability*) of sets of system states first. Two disjoint sets of states S_1 and S_2 are distinguishable iff

$$\forall s_1 \in S_1.\, \forall s_2 \in S_2.\, \exists \iota.\, \exists t.\, (Sys_S(s_1); Out^\dagger)(\iota)(t) \cap (Sys_S(s_2); Out^\dagger)(\iota)(t) = \emptyset$$

This means that for any two differing start states s_1 and s_2 from S_1 and S_2 there exists an input such that the observable output of the system starting in s_1 is disjoint from that of the system starting in s_2 after some finite time t. Note that disjointness is required to ensure that *all* nondeterministic choices which Sys_S can make for the two start states are different. Due to time invariance, observability provides that states from S_1 and S_2 can also be distinguished at any later time instant, not only when they are initial states. Finally, a system is called *fully observable* if every two distinct states can be distinguished, formally:

$$\forall s_1, s_2 \in \mathcal{S}.\, \exists \iota.\, \exists t \in \mathbb{R}_+.\, s_1 \neq s_2 \Rightarrow (Sys_S(s_1); Out^\dagger)(\iota)(t) \cap (Sys_S(s_2); Out^\dagger)(\iota)(t) = \emptyset$$

If a plant is not controllable or not observable we cannot design controllers which are able to drive it into every desired state, because either the desired state is unreachable

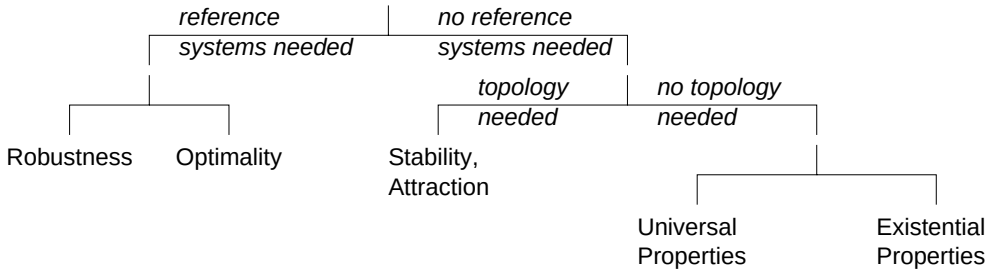


Figure 6.8: Classification of properties.

from the current state of the plant, or we cannot determine the current state due to lacking observability, or both. In this sense unobservability can be seen as a lack of (appropriate) sensor information, and uncontrollability can be interpreted as a lack of (appropriate) actuators. Such a lack may be caused by the underlying physics, for instance some quantities may be difficult to measure, but also by economic constraints, like too expensive actuators. Controllability and observability are central prerequisites for state-based design techniques of controllers [Oga87]. If applicable, such methods e.g. allow the synthesis of stable optimal controllers for certain purposes. However, controllability and observability have not been considered in any of our case studies.

In computer science properties of this kind can be expressed with *alternating-time temporal logic (ATL)* as defined in [AHK97]. In contrast to CTL [Eme90], ATL allows us to distinguish between the system and its environment in path quantification. Thus, we can express that for all possible moves of the system the environment can select moves (or inputs in our context) which ensure that a certain property holds.

6.3.6 Classification of the Properties and its Consequences

6.3.6.1 Classification

The classification of properties we propose is based on the semantic models relative to which the validity of the properties is defined.

By definition, the validity of robustness and optimality of a system must be determined relative to a set of reference systems (Fig. 6.8, left branch). For robustness, an evaluation function for the regarded object property is necessary additionally. Optimality instead requires a cost function. For the other properties in Section 6.3, no reference systems are needed to determine their validity (Fig. 6.8, right branch). Determining stability or attraction requires that topologies for the input and output space (or the state space, respectively) of the regarded system are given (Fig. 6.8, left branch at the second level). For the properties of Section 6.3.5, no topologies are necessary to determine their validity (Fig. 6.8, right branch at the second level). The properties of Section 6.3.5 are all evaluated in the same domain, namely w.r.t. a given system. As

Robustness	3
Optimality	2
Stability	6
Attraction	1
Universal properties	8
Existential properties	2

Table 6.4: Number of times a property was referenced at last once in a case study.

already indicated in the previous section, we partition this subclass further into the set of properties which constrain all behaviors of a system (universal properties) and the set of properties which demand existence of some specific behavior (existential properties). The existential properties can furthermore be divided into those demanding the existence of some behavior and those demanding the existence of causes enforcing certain effects (see Section 6.3.5.2).

6.3.6.2 Rating

In order to get a vague estimate of the relevance of the listed properties in practice, we count the number of case studies in which at least one property of a given kind is considered for each kind of property. For instance, if case study A considered three universal properties and two stability properties, the counter for the number of case studies considering universal properties and the counter for those considering stability properties are increased by one. Hence, the maximum possible count for a property class is nine, the number of all evaluated case studies. We do not use the *total* number of times a certain property class was considered for our rating, because this would assign inadequately high numbers to invariance properties, since a greater number of properties of this kind were listed in two of the case studies (in [BHKT98] and [Eng97]).

Table 6.4 lists the resulting rating. Although attraction is considered in only one case study, it probably is also relevant in the case studies which only considered stability, because in cases where stability is desired, attraction of the stable set usually is also desirable. The eight case studies regarding universal properties all regard invariance properties (Section 6.3.5.1). Two of them also regard an invariance property for a reduced set of causes $\mathcal{C}$. Namely, they require that for certain kinds of causes the system's state or output variables never exceed given values. In the two case studies which mention or examine an existential property, this regarded existential property is the existence of limit cycles.

Although the significance level of the given rating is low (only nine case studies were examined), there is a clear tendency indicating that universal properties, in particular invariance properties, and stability are most important. This clearly does not imply

that the other properties are irrelevant, but it indicates that it is sensible to put priority on work on methods for establishing these properties and maintaining them in transformations. Another possible conclusion is that these properties were regarded so frequently because the best analysis methods exist for them. In the author's opinion this can only be true in part. First, some of the regarded case studies merely explain a system and desirable properties without validating them; or validate some properties and describe others as also important, but do not analyze them. These mentionings of properties also appear in our rating. In this respect, the case studies only partially depend on existing methods. Second, the analysis method used most frequently in the case studies was simulation-based testing which can in principle be applied to all the properties. Nevertheless, note that from a theoretical point of view tests are not helpful w.r.t. unbounded liveness properties or for (infinite time) attraction, since in theory judging the correctness of an output would require infinite observation.

6.3.6.3 Consequences for Refinement

In this section, we want to consider which properties are maintained under *refinement*. The notion of refinement we employ was already introduced in the context of hierarchic graphs in Section 3.3.5 and is based on set inclusion. We say that relation A is a refinement of relation B, written as $A \preceq B$, iff $A \subseteq B$. Consequently, for I/O-based system descriptions this means that $Sys'_{IO} \preceq Sys_{IO}$ iff $\forall \iota \in \mathcal{I}^{\mathbb{R}+}. Sys'_{IO}(\iota) \subseteq Sys_{IO}(\iota)$. For state-based system descriptions, we have $Sys'_S \preceq Sys_S$ iff $\forall s \in \mathcal{S}, \iota \in \mathcal{I}^{\mathbb{R}+}. Sys'_S(s, \iota) \subseteq Sys_S(s, \iota)$. This expresses that the traces of the original system include those of the refined system, while both systems are required to be total in their input (and start states) because of our definition of systems (Section 6.2). Note that this notion of refinement is common in computer science. For instance, it is studied in [Bro99b, Bro97b].

From the definition of universal properties it is obvious that they are preserved under this notion of refinement. For stability and attraction, it is also easy to see that refinement maintains them. Simply note that for a refined system, we can choose the same neighborhoods of causes in a proof of stability or attraction as for the original system and end up with a subset of the traces of the original system for the same causes (and inputs, in the case of state-based system descriptions). Hence, traces in this subset are also contained in all the sets and neighborhoods of sets in which the original trace set is.

Similarly, controllability and observability are preserved under refinement. Informally, the reason is that they involve existential quantification over the input and (implicit) universal quantification over the output. As systems, whether refined or not, are required to be total in their input, inputs which provided controllability or observability for the original system also do so for the refined system with its smaller, but nonempty, set of traces for the same inputs.

Existential properties requiring the existence of effects, such as the existence of periodic behavior or limit cycles (Section 6.3.5.2), are not maintained by refinement. Again this is straightforward, because in the refined system exactly the "good verdict" may have been removed from the possible effects of the original system for given causes (and input).

The situation is more difficult for the properties robustness and optimality, because we also have to consider sets of reference systems, and object property and cost function respectively here. Let us first define what refinement means for reference systems. A set of reference systems R' is called a refinement of a set R of reference systems iff $\forall Sys' \in R'. \exists Sys \in R. Sys' \preceq Sys$. Thus, we require that for each system Sys' in the refined set there is a system in the original set which is refined by Sys'. Based on this definition, the following conclusion is an immediate consequence of the definition of robustness, because the definition involves universal quantification over the reference systems. Provided system Sys with deviations D, $Sys \in D$, robustly satisfies Ψ and provided that Ψ is preserved by refinement, then the refined system Sys' with refined deviations D', $Sys' \in D'$, also robustly satisfies Ψ. As we have seen above Ψ is preserved under refinement if, for instance, it is a universal property.

Optimality in general is not preserved by refinement. Here, the character of the cost function is crucial. For instance, a reasonable class of cost functions are functions which determine a system's quality as the supremum of the cost of all possible effects for given causes (and input). If we only refine the optimal system, but not the set of alternative systems, optimality is maintained, since the cost for the refined system could only have decreased while costs for the alternative systems remained equal. However, if the alternative systems are refined, optimality of a given system need not be preserved w.r.t. the new alternatives. Namely, there could be a refined alternative system from which expensive effects have been removed such that this system is now preferable to the old optimal system (or even to a refined version of it).

Contrasting these results with our rating of properties yields that the chosen refinement notion preserves most central properties. Merely existential properties with existential quantification over effects are not maintained. Moreover, for robustness and optimality the situation depends on the object property and the cost function, respectively.

6.4 Some Proof Concepts

In practice we have to distinguish between two kinds of applications of hybrid systems. First, there may be a control task that is realized by the hybrid system, e.g. guaranteeing that a certain physical quantity has a prescribed value. We call such a system a *control-centered hybrid system*. Second, the system's task may be to implement a certain process, e.g. to control the appropriate consecution of different process phases

such as heating material, shaping it and cooling it again. We call such a system a *process-centered hybrid system*. Note that this is not a sharp distinction. Control-centered hybrid systems may appear within larger process-centered systems and similarly, process-centered systems may be a part of control-centered systems. Robotic hand regrasping is an example for such a hierarchy [SHB+99]. There, the overall task is to guarantee that an object is held in a stable position by a robotic hand (control-centered aspect). A subtask consists of moving some fingers in order to obtain a better grasp (process-centered aspect).

For both kinds of systems, different classes of properties are important. For control-centered hybrid systems, stability and attraction are vital properties. In process-centered control systems, safety properties, like "buffer capacities are never exceeded", and liveness properties, like "the next phase is started eventually", play a major role. In this section, safety and liveness properties are not considered in greater detail because their underlying proof principles are largely familiar to computer scientists. Besides that, there already is significant work on this topic, e.g. [Lam93], where TLA (*Temporal Logic of Actions*) with its existing proof methods is extended to hybrid systems, or [Pnu94], where *computational induction* is introduced to prove invariance properties of hybrid systems. Note that in Appendix A.1.2 an inductive proof principle for HySCharts is defined which is similar to computational induction.

In the following, proof methods for state-based stability and state-based attraction of (sets of) points are presented. This kind of stability and attraction is most extensively studied in standard textbooks on control theory. Furthermore, we will develop specializations of the general methods which are helpful in applications and outline parallels to similar proof methods for discrete and continuous systems. In Section 6.5.1, the resulting methods are applied in a small example to prove stability and attraction of the EHC system. To emphasize parallels to computer science and demonstrate the applicability of the methods to discrete systems, a classic self-stabilizing system from computer science is considered as a further example in Section 6.5.2.

6.4.1 State-based Stability

Proofs of stability in control theory usually consist of finding a continuous[5] monotonously decreasing function from the system states to some assessment space, usually to the real numbers. This function is required to have a unique minimum for the state whose stability must be shown. According to the work of Liapunov existence of such a *Liapunov function* implies the stability of the unique minimum. In a physical interpretation Liapunov functions can often be regarded as energy functions. They associate the amount of energy in the system with each of its states. If energy is

[5]Unless otherwise mentioned, we use to the topological definition of continuity, not the one based on domain theory [SG90, Win93].

never added in the evolution of the control system, it is stable.[6] From a more abstract point of view, the Liapunov function can be seen as an abstraction that maps the real system to a qualitatively equivalent system [MW95].

The following theorem may be seen as a generalization of the classical Liapunov theory. It is adapted from [MT75]. In the theorem we write $\langle v \rangle$ for $\{v' \mid v' \sqsubseteq v\}$ for partial order $\sqsubseteq$ and $f^{-1}(y)$ for the inverse image of y under function f, i.e. $f^{-1}(y) = \{x \mid y = f(x)\}$. f^{-1} is extended to sets by pointwise extension. As the theorem is rather technical in part, it is explained in the following paragraph. Sections 6.4.1.1 and 6.4.1.2 introduce specializations which also help to understand the last two requirements of the theorem.

Theorem 6.1 *The set $A \subseteq \mathcal{S}$ is stable w.r.t. system $Sys_S \in \mathcal{S} \to (\mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{S}^{\mathbb{R}_+}))$, topology $\mathfrak{O}_S$, and inputs in $I \subseteq \mathcal{I}^{\mathbb{R}_+}$ if there exists a function $L \in \mathcal{S} \to V$ with:*

1. *V is a partially ordered set with partial order $\sqsubseteq$, and $V^+ \subseteq V$ is the subset of elements $v \in V$ for which there is a neighborhood of A such that the inverse image under L of all elements $v' \sqsubseteq v$ is not a proper subset of the neighborhood, formally $V^+ = \{v \in V \mid \exists \alpha \in N(A). \neg(L^{-1}(\langle v \rangle) \subsetneq \alpha)\}$.*

2. *L is monotonously decreasing along the traces of Sys_S, formally $\forall s \in \mathcal{S}, \iota \in I. \forall \sigma \in Sys_S(s, \iota). \forall t, t' \in \mathbb{R}_+. t' \geq t \Rightarrow L(\sigma(t')) \sqsubseteq L(\sigma(t))$.*

3. *$\forall v \in V^+. \exists \alpha \in N(A). \forall x \in \alpha. L(x) \sqsubseteq v$*

4. *$\forall \alpha \in N(A). \exists v \in V^+. \forall x. L(x) \sqsubseteq v \Rightarrow x \in \alpha$*

Proof. Let $\alpha \in N(A)$ be an arbitrary neighborhood of A. Application of 4 and 3 yields that there is a $\beta \in N(A)$ and a $v \in V^+$ with $x \in \beta \Rightarrow L(x) \sqsubseteq v$, and therefore $x \in \alpha$. For $x \in \beta$ and $\sigma \in Sys_S(x, \iota)$ monotonicity yields $\forall t. L(\sigma(t)) \sqsubseteq v$. Hence, $\sigma \in \alpha^{\mathbb{R}_+}$. □

If existing, such a function L is called a *Liapunov function*. Informally the combination of the last two requirements expresses that for any neighborhood of A there exists a smaller neighborhood whose L-image is bounded from above by some $v \in V^+$. In terms of the inverse images under L the last two requirements can equivalently be written as $\forall v \in V^+. \exists \alpha \in N(A). \alpha \subseteq L^{-1}(\langle v \rangle)$ and $\forall \alpha \in N(A). \exists v \in V^+. L^{-1}(\langle v \rangle) \subseteq \alpha$, respectively. The set V^+ eliminates all those elements from V which are not helpful in the proof of stability, because they constrain the considered sets of points too strongly, namely to the inside of all neighborhoods of A. It is needed to simplify the application of the theorem, because specializations of the theorem usually use sets V with bottom element $\perp$ and mappings L with $L^{-1}(\perp) = A$. If the universal

[6] Adding energy does not violate the physical law of energy preservation, because the control system may use external sources for increasing the energy.

quantification in requirement three ranged over all $v \in V$, it would not be satisfiable for these specializations in topologies on $\mathcal{S}$ in which A is a closed set.

Note that the above theorem has some parallel to the proof method given in [Sin92] to prove a property called *atomicity of an invariant* for a given dynamical system. This property expresses that a set which is invariant for the given system (Definition 6.5) is a singleton. The parallel to Theorem 6.1 is due to both proof methods using a comparison function L to show that the considered system has a non-expansive (in the case of stability) or contractive character (in the case of atomicity).

In the following we consider two specializations which are helpful for applications and which make parallels to the notion of abstraction in computer science explicit.

6.4.1.1 Liapunov Functions and Galois Connections

The last two requirements in Theorem 6.1 suggest that there are two total mappings from V^+ to the neighborhoods of A (Requirement 3) and back again (Requirement 4). Such mappings are similar to abstraction via Galois connections [CC92], which is a common technique in computer science. Informally, a Galois connection is a pair of monotonous mappings, an *abstraction function* and a *concretization function*, from a "concrete" partially ordered set to an "abstract" partially ordered set and back again. The compositions of these mappings are required to loose information in a way consistent with the regarded partial orders. We will make the similarity between Liapunov functions and Galois connections explicit now. This idea leads to a specialization of Theorem 6.1 which is developed in the following.

Assumptions. Let $(V, \sqsubseteq)$ be a complete partial order which furthermore is densely ordered by the strict version $\sqsubset$ of $\sqsubseteq$ and has a least element $\bot$ (see Appendix B.1 for the definition of densely ordered sets). Let L be a mapping from $\mathcal{S}$ to V such that the given topology on $\mathcal{S}$ is the coarsest topology which makes L continuous w.r.t. the interval topology induced by $\sqsubseteq$ on V (see Appendix B.2 for the definition of interval topology). L is required to be onto and the L image of all elements in A must be $\bot$. Furthermore, V and L must be chosen such that $V^+ = V \setminus \{\bot\}$, where V^+ is defined as in Theorem 6.1.[7] We define function $abs \in N(A) \to V^+$ by $abs(\alpha) = sup\{v \in V^+ \mid \alpha \supseteq L^{-1}(\langle v \rangle_o)\}$, where $\langle v \rangle_o$ is the set of all those v' which are strictly "less than" v, $\langle v \rangle_o = \{v' \mid v' \sqsubset v\}$. Function $conc \in V^+ \to N(A)$ is given by $conc(v) = L^{-1}(\langle v \rangle_o)$. Lemma A.16 in Appendix A.3 proves that abs and $conc$ are well-defined.

Theorem 6.2 *Functions abs and conc are a* Galois connection *between the spaces* $(N(A), \supseteq)$ *and* $(V^+, \sqsupseteq)$*, where* $\sqsupseteq$ *is given by* $v \sqsupseteq v' :\Leftrightarrow v' \sqsubseteq v$.[8]

[7] This can be achieved by choosing L such that $L^{-1}(\bot) = A$.

[8] Actually *abs* and *conc* make up a *dual* Galois connection [CC92], because in the usual terminology of [NNH98] and [CC92] our *abs* is the concretization and *conc* the abstraction.

By the definition of Galois connections [CC92] this means that monotonicity of *abs* and *conc*, and $\alpha \supseteq conc(abs(\alpha))$ (*extensivity*) and $abs(conc(v)) \sqsupseteq v$ (*reductivity*) must be proven. This in done in Theorem A.17 of Appendix A.3. In fact, a stronger variant of reductivity, $abs(conc(v)) = v$, is proven there, which implies that *abs* and *conc* even are a *Galois surjection* on the considered spaces.

Specialization for Stability. The given definition of *abs* and *conc* allows us to derive stability of A from the monotonicity of L w.r.t. the system traces of Sys_S. The proof is given in Appendix A.3, Theorem A.18. This amounts to a specialization of Theorem 6.1, as a L function with the properties described in the assumptions above also satisfies the requirements of that theorem. Namely, *abs* and *conc*, which are defined depending on L, immediately help to satisfy Requirements 3 and 4 of Theorem 6.1.

Note that monotonicity of L can also be interpreted as stability of $\bot$ w.r.t. the L-image of Sys_S in V. We will consider this in more detail in Section 6.4.1.2. Furthermore, note that the stability proof also works if V is not complete and densely ordered, but just a partially ordered set with bottom element. In this case the supremum operator, *sup*, in the definition of *abs* must be replaced by Hilbert's non-deterministic choice operator and *abs* no longer needs to be monotonic w.r.t. $\supseteq$. Hence, *abs* and *conc* no longer build a Galois connection. Besides that, to proof stability L need not be onto. This is only necessary to show that *abs* and *conc* are a Galois connection.

Using the interval topology on V together with the other requirements we stated for V and L implies that $L^{-1}(\bot)$ is a closed set in $\mathcal{S}$, because $\{\bot\}$ is closed in V. If we choose L such that $L^{-1}(\bot) = A$, this suits well to standard control theory books. There, Liapunov functions usually use $V = \mathbb{R}_+$, the considered set A is a singleton, and therefore closed w.r.t. the natural topology on the Euclidean space, and $L(x)$ has its unique minimum for $\{x\} = A$.

Thinking in terms of abstraction and concretization functions can help us to find Liapunov functions. Namely, they lead our intuition to looking for equivalence classes in $\mathcal{S}$. The elements of an equivalence classe are then defined to produce the same L-value. In our view it is more constructive to search for a continuous L function such that the above *abs* is well-defined than to find a function satisfying the last two requirements of Theorem 6.1, since these requirements are very abstract.

6.4.1.2 Liapunov Functions and Homeomorphisms

A further specialization of Theorem 6.1 results if we demand that L be a *homeomorphism*, i.e. a function which is one-to-one and onto and has a continuous inverse L^{-1}. In detail the assumptions are as follows.

Assumptions. Let V be partially ordered by $\sqsubseteq$ and have a least element $\bot$. Let $L \in \mathcal{S} \to V$ be a homeomorphism, where continuity of L^{-1} is required w.r.t. the

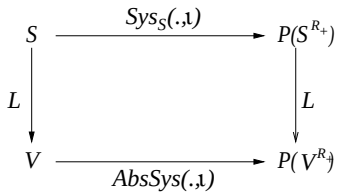


Figure 6.9: Commuting diagram for L.

topology on $\mathcal{S}$ and the left topology on V (see Appendix B.2 for the definition of left topology). This implies that L also is continuous w.r.t. the same topologies [Eng89]. Furthermore, L must be monotonous w.r.t. the traces of Sys_S (see requirement two in Theorem 6.1), and it must map all elements in A to $\perp$. Note that together with L being one-to-one and onto this implies that A may only contain one element. For V^+ defined as in Theorem 6.1 the above requirements for V and L imply that $V^+ = V$ holds, since $\langle\perp\rangle$ is an open set w.r.t. the left topology and $L^{-1}(\langle\perp\rangle) \in N(A)$ therefore is the smallest neighborhood of A.

Remark that in contrast to the previous paragraph we here employ the left topology on set V instead of the interval topology. We do so in order to give an example of a class of L functions for which $L^{-1}(\perp)$ is an open set, which suits well in cases where A is open and $L^{-1}(\perp) = A$. Nevertheless, the arguments presented here can easily be adapted to totally ordered sets V with the interval topology.

Specialization for Stability. Theorem A.19 in Appendix A.3 shows that the above assumptions imply the requirements of Theorem 6.1. As a consequence, they are sufficient to conclude the stability of A w.r.t. Sys_S and the topology on $\mathcal{S}$. This gives us a specialization of Theorem 6.1.

Abstraction. From a computer science perspective, L can be seen as an abstraction mapping the *concrete* system Sys_S to an *abstract* system $AbsSys \in V \to (\mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(V^{\mathbb{R}_+}))$ such that (1) L *commutes* between Sys_S and $AbsSys$ (see Figure 6.9), and (2) L preserves topological properties on $\mathcal{S}$ and V. In [Aki93] such a mapping is called a *(topological) conjugacy*. Here, the abstract system $AbsSys$ is defined as $AbsSys(v, \iota) = L(Sys_S(L^{-1}(v), \iota))$, where L is extended to trajectories and sets of trajectories in a pointwise manner. L commutes between Sys_S and $AbsSys$, i.e. $L(Sys_S(s, \iota)) = AbsSys(L(s), \iota)$, because of the definition of $AbsSys$ and because L is one-to-one and onto. In our context, stability of $L(A)$ for the abstract system $AbsSys$ implies stability of A for the concrete system Sys_S. Due to using the left topology on V, stability of $L(A)$ is a consequence of monotonicity of L w.r.t. the traces of Sys_S. To see this, note that stability of $L(A) = \{\perp\}$ can be written as $\forall v.\, \exists v'.\, \forall \iota \in I.\, \forall v'' \sqsubseteq v'.\, \forall \eta \in Abs(v'', \iota).\, \forall t.\, \eta(t) \sqsubseteq v$, where we implicitly associate the neighborhood $\langle v\rangle$ of $\{\perp\}$ with every $v \in V$. By the definition of $AbsSys$ this formula for stability of $\{\perp\}$ is a direct consequence of the monotonicity of L along the traces of Sys_S (requirement 2 of Theorem 6.1).

We remark that, as stability depends on topology, it is not surprising that studying stability properties by abstraction requires to also relate the topologies on the abstract and the concrete space. This is done by a homeomorphism and also by the particular Galois connection given above. In case of the homeomorphism, continuity of L and L^{-1} establishes the needed relationship between open sets in both spaces. In the case considered in Section 6.4.1.1, continuity of L establishes the connection from open sets in the abstract space to open sets in the concrete space. Requiring that the topology on the concrete space is the coarsest which makes L continuous establishes the other direction.

Note that finding a Liapunov function which defines a Galois connection as explained in Section 6.4.1.1 can be regarded as a greater abstraction than finding a homeomorphism. This holds true, because in the case of a homeomorphism the concrete space $\mathcal{S}$ and the abstract space V are isomorphic. In contrast, a L function that (only) leads to a Galois connection allows us to identify elements of the concrete space in the abstract space. Hence, information may be lost in the latter case, while no loss occurs in the former case.

6.4.2 Attraction

Control theory usually considers attraction together with stability, i.e. asymptotic stability. Like in Section 6.4.1 Liapunov functions are also used in order to proof asymptotic stability. However, for asymptotic stability the functions must obey additional restrictions which ensure that the Liapunov function indeed reaches a certain unique minimum. For continuous time systems, a sufficient criterion is that L is continuously differentiable along the system trajectories with the derivative being strictly less than zero everywhere except of at the minimum [Lue79].

In our general framework, where attraction does not necessarily imply stability, the following theorem allows us to proof attraction of points for state-based systems. It corresponds to Theorem 6.1 with a different third requirement, which is explained informally below.

Theorem 6.3 *The set $A \subseteq \mathcal{S}$ is globally attractive w.r.t. system $Sys_S \in \mathcal{S} \to (\mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{S}^{\mathbb{R}_+}))$, the topology $\mathfrak{O}_\mathcal{S}$ and the inputs in $I \subseteq \mathcal{I}^{\mathbb{R}_+}$ if there exists a function $L \in \mathcal{S} \to V$ with:*

1. *V is a partially ordered set with partial order $\sqsubseteq$, and $V^+ \subseteq V$ is the subset of elements $v \in V$ for which there is a neighborhood of A such that the inverse image under L of all elements $v' \sqsubseteq v$ is not a proper subset of the neighborhood, formally $V^+ = \{v \in V \mid \exists \alpha \in N(A). \neg(L^{-1}(\langle v \rangle) \subsetneqq \alpha)\}$.*

2. *L is monotonously decreasing along the traces of Sys_S, formally $\forall s \in \mathcal{S}, \iota \in I. \forall \sigma \in Sys_S(s, \iota). \forall t, t' \in \mathbb{R}_+ . t' \geq t \Rightarrow L(\sigma(t')) \sqsubseteq L(\sigma(t))$.*

3. $\forall v \in V^+, s \in \mathcal{S}, \iota \in I. \forall \sigma \in Sys_S(s,\iota). \exists t. L(\sigma(t)) \sqsubseteq v.$

4. $\forall \alpha \in N(A). \exists v \in V^+. \forall x. L(x) \sqsubseteq v \Rightarrow x \in \alpha.$

Proof. Let $\alpha \in N(A)$ be an arbitrary neighborhood of A and $s \in \mathcal{S}$, $\iota \in I$ arbitrary. Using 4 to select a v for α such that states with L-values less or equal v are in α, and applying 3 yields that for all $\sigma \in Sys_S(s,\iota)$ there is a t such that $L(\sigma(t)) \sqsubseteq v$. Monotonicity yields $L(\sigma(t')) \sqsubseteq v$ for all $t' \geq t$ and hence, due to the choice of v, $\sigma(t') \in \alpha$ for all $t' \geq t$. □

Similar to Theorem 6.1, V^+ is needed here, because otherwise the theorem would exclude the frequently occurring case where V contains a bottom element $\bot$, L is such that $L^{-1}(\bot) = A$, and A is a closed set. Requirement three expresses that any $v \in V^+$ is eventually reached by all trajectories of Sys_S. Obviously, the existence of a Liapunov function $L \in \mathcal{S} \to V$, as defined by Theorem 6.1, which also satisfies requirement three of Theorem 6.3, proves asymptotic stability of the considered set A. Hence, we can also use Galois connections of the kind defined in Section 6.4.1.1 or homeomorphisms as defined in Section 6.4.1.2 to prove attraction, provided they also satisfy requirement three from above.

Theorem 6.3, in general, does not imply Theorem 6.1. Informally the reason is that for attraction a function L suffices which, for every trace, is constant in the beginning and converging to a specific minimum as time goes to infinity. If the chosen function L does not fit to the topology on $\mathcal{S}$ in the way demanded by the third requirement of Theorem 6.1, this constant value on the beginning of a trace need not yield any information on the neighborhood of A in which the trace is at the respective time instants.

Requirements two and three of the theorem together yield that $\forall v \in V^+, \iota \in I, s \in \mathcal{S}. \forall \sigma \in Sys_S(s,\iota). \exists t. \forall t' \geq t. L(\sigma(t')) \sqsubseteq v$ holds, which exactly is what is needed in the proof. If V has a least element $\bot$ and $V^+ = V$ this property is equivalent to global attraction of $\{\bot\}$ w.r.t. the left topology on V induced by $\sqsubseteq$ (see Appendix B.2 for the definition of the left topology).[9] This underlines that, just as for stability, L is an abstraction mapping which allows us to carry over attraction in V to attraction in $\mathcal{S}$.

6.4.2.1 Specializations and Parallels to Discrete and Continuous Systems

A problem for the practical utility of Theorem 6.3 is that requirement three is not very handy. In fact, convergence, which is expressed in this requirement, usually is difficult to prove. This, however, can be alleviated by selecting a set V for which there is a rich theory for convergence. We therefore want to consider a few specializations here.

[9] A similar result holds for the interval topology.

Discrete systems. If a discrete-time or discrete-event system is regarded, and if A is open w.r.t. the topology on $\mathcal{S}$, Theorem 6.3 can be specialized as follows. We choose V and $\sqsubseteq$ such that $\sqsubseteq$ is a well-founded order on V and V has a bottom element, e.g. $V = \mathbb{N}$ with bottom element $\bot = 0$ and $\sqsubseteq$ the less-or-equal relation $\leq$. The inverse image of $\bot$ under L must be A which implies $V^+ = V$, since A is open. We then require that for every trace there is an infinite, strictly monotonously increasing sequence of time instants $t_i \in \mathbb{R}_+$ at which the system performs its moves. L must be monotonously decreasing along the traces of Sys_S and strictly decreasing at every t_i unless the L-images of the states reach $\bot$ (from then on monotonicity suffices). Our traces σ are infinite and V only contains finite (strictly) decreasing sequences, because $\sqsubseteq$ is well-founded. Therefore, it follows that $L(\sigma)$ becomes constantly $\bot$ starting from some time t, i.e. convergence of the L-image of Sys_S to $\bot$ is ensured. Hence, these requirements imply requirements two and three of Theorem 6.3. Requirement four is a consequence of $L^{-1}(\bot) = A$, because for any neighborhood of A we can choose $v = \bot$ to satisfy the requirement. Consequently, this is a specialization of Theorem 6.3. It is applied in the example of Section 6.5.2. Note that this kind of L functions correspond to Floyd functions used in computer science to prove liveness properties [Cou90, Flo67], as [Sin92] mentions in a discrete time context. However, in comparison to Floyd functions we also require that L remains non-increasing when set A has been reached. This is needed to also proof the safety part contained in a persistence property.[10] In the example of Section 6.5.2 this becomes apparent.

Continuous-time systems. Classical strictly monotonous Liapunov functions are a second special case of the above theorem [Lue79]. Here, the considered set A is a singleton and therefore closed w.r.t. the natural topology on the Euclidean space used for space $\mathcal{S}$. $\mathbb{R}_+$ is chosen for V, with $\bot = 0$. For $\sqsubseteq$ the less or equal relation $\leq$ is used, and $V^+ = V \backslash \bot$. L is required to be continuously differentiable along the traces of Sys_S with continuous derivative $\frac{d}{dt}(L(\sigma(t))) < 0$ for all $\sigma(t) \notin A$ and $\frac{d}{dt}(L(\sigma(t))) = 0$ for $\sigma(t) \in A$.[11] Furthermore, $L(s)$ must be 0 iff $s \in A$, i.e. $L^{-1}(0) = A$. Convergence of L to 0 is a consequence of the strict monotonicity of L *and* the continuity of $\frac{d}{dt}(L(\sigma(t)))$. Strict monotonicity alone only implies convergence to *some* value. Requirement four is satisfied for L, because of continuity of L by topological arguments roughly similar to those given in Section 6.4.1.1.

Specialization with uniform monotonicity. For topologies in which A is not open, a similar more general specialization of the above theorem is obtained as follows. We again use $V = \mathbb{R}_+$ with bottom element and order as above, and require a kind of uniform monotonicity of L along the traces of Sys_S. This specialization is an adaption of the criterion given in [SG95] for attraction for discrete-time systems. In detail, L

[10] Remember that attraction is a persistence property from a computer science point of view (Section 6.3.4.2).

[11] Note that the derivative $\frac{d}{dt}(L(\sigma(t)))$ can also be written as $\frac{dL(s)}{ds} \cdot \frac{d\sigma(t)}{dt}$ under appropriate restrictions.

must be chosen such that $L^{-1}(0) = A$ and $V^+ = V \setminus \{0\}$ hold. Then, requirements two and three of Theorem 6.3 are replaced by demanding that for all $s \in \mathcal{S}$, $\iota \in I$ and $\sigma \in Sys_S(s, \iota)$ there is a $k \in [0, 1)$ such that $\forall t, t' \in \mathbb{R}_+ . t' \geq t \Rightarrow L(\sigma(t')) \leq k^{t'-t} \cdot L(\sigma(t))$. Informally this means that $L(\sigma(t))$ is required to decline at least as fast as k^t. Obviously, this implies monotonicity of L along the traces of Sys_S (second requirement of Theorem 6.3). To see that any $v \in V^+$ is reached or underpassed by any trace (third requirement of Theorem 6.3), let $v \in V^+$ be arbitrary and let $\sigma \in Sys_S(s, \iota)$ be a trace for start state s and input $\iota \in I$. Due to the above claim, there is a $k \in [0, 1)$ such that $\forall t. L(\sigma(t)) \leq k^t L(\sigma(0))$. For $t \to \infty$, $k^t L(\sigma(0))$ converges to 0 which implies that $L(\sigma(t))$ reaches v or falls below v eventually. Thus, with requirement four of Theorem 6.3 remaining unchanged, this yields a sound specialization of Theorem 6.3.

6.5 Applications

6.5.1 Stability and Attraction for the EHC

As example we consider the stability and attraction of the tolerance interval of the EHC system for disturbances of the system's initial state and in presence of bends. We use the abstract model of the EHC system as described informally in Section 6.1. Table 6.5 defines the actions, invariants and activities occurring in the HySChart given in that section (Figure 6.2). Once stability and attraction is shown for this abstract system, the HySChart can be refined further using techniques presented in preceding chapters. In particular this includes refining the system into the controller and suspension's dynamics, and discretizing the controller.

The primary aim of the EHC is to compensate different load situations of a car. At the considered stage it is not supposed to actively influence driving dynamics by, e.g., adjusting the suspension to different speeds or lateral accelerations. In fact it is explicitly required that the actuators be turned off when the car is going through bends. Because of control being suspended in state *inBend* no region A around the tolerance interval can be stable or attractive for *all* kinds of disturbances. If A (or a given neighborhood of it) has been reached, a curve may be entered. Then, the disturbances occurring in the curve may cause that the chassis level is outside A (or the regarded neighborhood) when the bend ends. This contradicts attraction. Stability is violated in a similar way. Nevertheless, this behavior of the EHC is acceptable if after a disturbance and despite of bends with finite duration the chassis level does not diverge further, but approaches the tolerance interval again. We summarize this by requiring that set A, whose formal definition we give in the following paragraph, is stable and globally attractive w.r.t. the system of Figure 6.2 and w.r.t. the set of inputs I. Set I is defined such that it expresses that bends are entered and left infinitely often with entries (exists) and exits (entries) of bends separated from each other by at least time bs. Moreover, for inputs in I the chassis level is never influenced

Definition of actions, invariants, and activities:

$b2o$	$\equiv$	$bend?$
$o2b$	$\equiv$	$b2o$
$i2u$	$\equiv$	$height \leq lb$
$i2d$	$\equiv$	$height \geq ub$
$u2i$	$\equiv$	$height \geq lb + c$
$d2i$	$\equiv$	$height \leq ub - c$
$CtrlSys_{inv}$	$\equiv$	$True$
$inBend_{inv}$	$\equiv$	$bend.t = bend.t^{\backprime} \ \vee \ now - bend.t^{\backprime} < \varepsilon_{bend}$
$outBend_{inv}$	$\equiv$	$inBend_{inv}$
$inTol_{inv}$	$\equiv$	$height \in (lb - \varepsilon_{height}, ub + \varepsilon_{height})$
up_{inv}	$\equiv$	$height < lb + c + \varepsilon_{height}$
$down_{inv}$	$\equiv$	$height > ub - c - \varepsilon_{height}$
$const$	$\equiv$	$\dot{a} = 0 \ \wedge \ \frac{d}{dt} height = \dot{a} + dist$
inc	$\equiv$	$\dot{a} \in [cp_-, cp_+] \ \wedge \ \frac{d}{dt} height = \dot{a} + dist$
dec	$\equiv$	$\dot{a} \in [ev_-, ev_+] \ \wedge \ \frac{d}{dt} height = \dot{a} + dist$

Table 6.5: Definition of actions, invariants, and activities of *CtrlSys* in Figure 6.2. (lb, ub, cp_-, cp_+, ev_-, ev_+ , c, ε_{height} and ε_{bend} are constants.)

from outside. Hence, we only consider initial disturbances of the chassis level. As the system is time invariant, this ensures that whenever the chassis level is outside the tolerance interval it approaches the tolerance interval again, provided bends satisfy the claimed assumptions. Note that we furthermore require that $bs > \varepsilon_{bend}$ holds, i.e. the time between entry and exit, and exit and entry respectively, of a bend must be greater than the maximum delay ε_{bend} with which the system reacts to them.

Definitions. Formally, I is defined as follows:

$$
\begin{aligned}
I \ = \ \{(be, di) \in (\mathbb{R}_+ \times \mathbb{R})^{\mathbb{R}_{p+}} \ | \\
\forall t \in \mathbb{R}_+ . \, di(t) = 0 \ \wedge \\
(t = 0 \ \vee \ \lim_{x \nearrow t} be(x) \neq be(t)) \Rightarrow \\
\exists t' \geq t + bs . \, be|_{[t,t')} = t^{\dagger}|_{[t,t')} \ \wedge \ be(t') = t' \ \}
\end{aligned}
$$

where $t^{\dagger}$ denotes the constant function mapping all arguments to t. For $(be, di) \in I$ the first component of the tuple denotes the evolution of the input on channel *bend* and the second that on channel *dist*. The definition provides that the input on channel *dist*, denoting external disturbances of the chassis level, is constantly 0, i.e. there is no disturbance. Moreover, the definition expresses that the time stamps on channel *bend*, whose values denote the last time of entry or exit of a bend, change infinitely often with two consecutive changes separated at least by time bs.[12] Note that set I could also have

[12]The definition also implies the time stamps are correct (see Example 5.4 in Section 5.4.1).

been defined with some hybrid temporal logic, HySCs [GKS00], or HySCharts. The proposed attractive and stable set A is $A = \{s \in \mathcal{S} \mid s_{height} \in [lb - \varepsilon_{height}, ub + \varepsilon_{height}]\}$, where we write s_{height} for the value of variable *height* in state s (see Figure 6.10 for a visualization of the interval and the relevant constants). As the detection of events is required in the example, the state space $\mathcal{S}$ we allow to be disturbed is not the whole state space of *CtrlSys*, but only consists of the control-state space and the ranges of the variables *height* and a. Variable *now* and the latched time stamp for the input channel *bend* are initialized such that they are consistent with the time stamps in the set of inputs I. This means that they are 0 in all considered initial states. The regarded topology on $\mathcal{S}$ is constructed as follows. For variable *height* we use the topology generated by the base $\mathcal{B}_h = \{\{x \mid d(x, A) \in b\} \mid b \in \mathcal{B}_{\mathbb{R}_+}\}$, where $\mathcal{B}_{\mathbb{R}_+}$ is a base for the natural topology on $\mathbb{R}_+$ and $d(x, A) = inf\{|x - a| \mid a \in A\}$ (Euclidean distance). The resulting topology is similar to the natural topology on $\mathbb{R}$, but coarser. It does not distinguish between elements of A and between deviations from A to higher or to lower values. This is consistent with our view of the EHC system, because all values of *height* inside set A are regarded as desirable, while too high and too low values of *height* are likewise undesirable. The variable a as well as the control-state space are immaterial here and we therefore use the coarsest topology for them, i.e. for variable a, for instance, we use the topology $\mathfrak{O} = \{\emptyset, \mathbb{R}_+\}$. The topology $\mathfrak{O}_\mathcal{S}$ we use on $\mathcal{S}$ is the Tychonoff (or product) topology of these topologies (see Appendix B.2, Definition B.10).

Stability. To prove stability of set A we use a Liapunov function of the kind described in Section 6.4.1.1. $L \in \mathcal{S} \to V$ is defined by $L(s) = d(s_{height}, A)$, i.e. it is the distance of *height* to A. We use $V = \mathbb{R}_+$ with order $\leq$ on the reals. V is complete and densely ordered, has least element 0, and $V^+ = V \setminus \{0\}$ (see Theorem 6.1 for the definition of V^+). L is onto, and $L(s) = 0$ if $s_{height} \in A$. The interval topology on V is equal to the natural topology on $\mathbb{R}_+$. Applying Theorem B.3 of Appendix B.2 proves that topology $\mathfrak{O}_\mathcal{S}$ is the coarsest topology on $\mathcal{S}$ which makes L continuous. In fact, this by part motivated our choice of the topology for the chassis level.

Using the result of Section 6.4.1.1 stability of A can now be established by proving that L is monotonously decreasing along the traces of *CtrlSys* for all inputs in I. The proof proceeds by trace induction over the traces of *CtrlSys*, as defined in Appendix A.1.2, Theorem A.6. Here, we argue informally. For a state $s \in \mathcal{S}$ at time t, *CtrlSys* either performs a discrete transition, or time passes until time $t' > t$. As no transition modifies the value of s_{height}, L remains constant, and hence monotonous, whenever a transition is taken. If control is in state *inTol* or in *inBend* and time passes, s_{height} also remains constant. If control is in *up* and time passes, its invariant, $height < lb + c + \varepsilon_{height}$, must hold and *height* is increased ($0 < cp_- < cp_+$). Thus, provided the constants are chosen such that $lb + c + \varepsilon_{height} < ub + \varepsilon_{height}$, L decreases until *height* reaches $lb - \varepsilon_{height}$ and then remains constant, since *height* then already is in A. If control is in *up* and the invariant does not hold, i.e. *height* already is too high, no time can pass in the state and a transition is taken. The situation is similar for

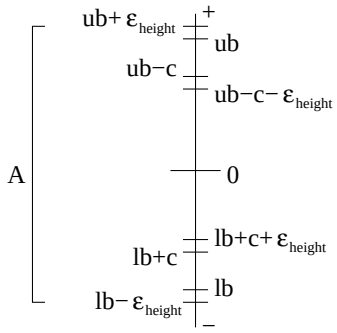


Figure 6.10: Relevant intervals and constants for the chassis level.

control being in *down*. Hence, L is monotonous.

Attraction. For attraction we use the same sets I, V, V^+ and function L as defined above. We have to prove that every $v \in V^+$ is reached (requirement 3 in Theorem 6.3). The other requirements of the theorem are already satisfied for L, because L was chosen according to the specialization of Theorem 6.1 given in Section 6.4.1.1. Again, we argue informally. Let $s \in \mathcal{S}$ be some state. If s is in A nothing remains to be shown as $L(s) \leq v$ already holds. If in the regarded state the height is below $lb - \varepsilon_{height}$, there are two possibilities depending on whether the control state is in *inBend* or in one of the substates of *outBend*. In the first case, some time will pass with *height* remaining constant. Then, at most time ε_{bend} after the next toggling of *bend* in the input, control changes to *inTol* and we have the second case. In the second case, the control state immediately changes to *up*, because the transitions leading to *up* are enabled and the invariants of *inTol* and *down* are false. Control will remain in *up* until either the height is in A or a bend occurs. In case of a bend, the minimal separation bs between *bend* signals in the input ensures that time $bs - \varepsilon_{bend}$ passes in *up* (or A is reached) before state *inBend* is entered.[13] Note that we required that $bs > \varepsilon_{bend}$ above. As long as control is in *up* the height decreases at least with rate cp_-. Thus, between two bends the height is always decreased by at least $cp_- \cdot (bs - \varepsilon_{bend})$ in *up* unless the height is in A. As bends are left infinitely often, the regarded v is reached after finitely many curves, and therefore after a finite amount of time. Note that the height, of course, also is decreased in *up* if no bends occur. Finally, if in the regarded state the height is above $up + \varepsilon_{height}$, the argument is analogous to the above case. This ends the proof, by Theorem 6.3 we obtain that A is attractive.

Remarks. We could also have incorporated the model of disturbances into the system model and could have examined the resulting (input-free) model. For complex disturbances for hybrid systems this may often be necessary for proofs, as certain control states can only occur under certain environment behavior. For the EHC, e.g.,

[13] Only time $bs - \varepsilon_{bend}$ not time bs is guaranteed to pass in *up*, because the reaction to the *bend* signal may be delayed by at most ε_{bend}.

leaving a bend is only possible if a bend has been entered. Here, not incorporating the inputs into the state forced us to consider infinitely many bend events only because we cannot decide which event models leaving a bend and which denotes entry.

Furthermore, we could also use a control-state depended Liapunov function. This has not been needed here, but it would be necessary if we allowed (small) disturbances inside and outside bends (where disturbances inside bends are required to vanish).

Proofs of stability and attraction are also possible for more general models of external disturbances. However, such proofs require a lot of technical detail. We rather decided to give a simple example here to demonstrate proofs of stability and attraction for hybrid systems.

6.5.2 Attraction for Self-Stabilizing Algorithms

The example in this section shows how a correctness proof for a so called self-stabilizing algorithm can be modified to result in a proof of attraction. Due to the used topologies attraction will turn out to be equivalent to the original safety- and liveness properties subsumed in self-stabilization. Thus, this example shows the relevance of attraction also for computer science.

In [Dij82], Dijkstra defines *self-stabilizing systems* as systems which are guaranteed to arrive at a legitimate state after a finite number of steps regardless of their initial state. Furthermore, once a legitimate state has been reached, the system is required to remain in such legitimate states forever. Note that this term may not be confused with the definitions of stability we have given here. In our terms self-stabilization rather is related to attraction than to stability. In terms of [CMP91], self-stabilization is a persistence property. Similar to attraction, self-stabilization can be useful as a substitute for an initialization procedure or for error recovery. References on self-stabilizing systems can e.g. be found in [Ali99].

The system. The system we consider here is taken from [Dij82]. It consists of a ring of $N+1$ processes, $N > 0$, numbered from 0 to N in clockwise direction. Each process has a variable with a value in $\{0, \ldots, M\}$, where $M \geq N$. A process with number $(i+1)$ *mod* $(N+1)$ may read and write its own variable and read the variable of its left-hand neighbor i. (Note that neighbors in counter clockwise direction are also called *left-hand* neighbors.) The legitimate states of the system are those where there is exactly one *token* (see below) in the ring.

The system evolves as follows. Process 0 is said to hold a token iff its variable *is equal* to that of its left-hand neighbor. When holding the token it can make a step and increment its variable by 1 modulo $M+1$. Every other process is said to hold a token iff its variable *is different* from that of its left-hand neighbor. In this case it may make a step and set its variable to the value of his left-hand neighbor's variable. The process which actually executes its step is selected by a fair nondeterministic scheduler. In the

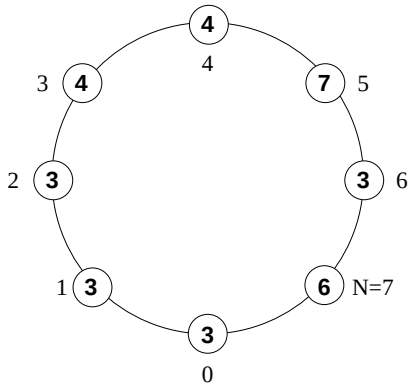


Figure 6.11: An example configuration for N=7 and M=7.

example of Figure 6.11 process numbers are written besides the circles which denote processes. The values of their variables are written in bold font inside the circles. In the depicted configuration processes $3, 5, 6$ and 7 are *enabled*, i.e. they hold a token.

Dijkstra proves that for any initial state, the system eventually reaches a legitimate state, i.e. a state with only one token in the ring, and then remains within the set of legitimate states forever [Dij82]. We will follow the more formal proof in [Mer98] which uses the standard technique for proving liveness properties. This technique consists of defining a valuation function from the state space to a set with a well-founded order such that the valuation function decreases with every step of the system. Furthermore, an infinite number of steps is assumed to exist. As we will see, after a slight modification this valuation function can be used as a Liapunov function that proves the attraction of the set of legitimate states. In order to remain within the continuous-time system model used throughout this chapter we embed the algorithm in continuous-time by using a modified scheduler which schedules the next process to proceed at a nondeterministically selected finite time after the current step. During the intervals between the selected time instances all processes are idle; the system state remains constant.

Solution principle. Informally the algorithm works as follows. The new values created by process 0 spread out from this process in clockwise direction, i.e. there is a "prefix" of processes in the ring having the same value as process 0. The values held by the variables of processes outside the prefix are potentially "bad", because they might provide tokens to too many processes. The legitimate states are those, where there either is only one value outside the prefix (then the process immediately following the prefix in clockwise direction is enabled), or the prefix is the whole ring. In this case process 0 is enabled. Values different from the one in the prefix are eliminated as time elapses. They vanish at process N, because process 0 creates no new value for them. If the value in the prefix also occurs in one of the processes outside the prefix, process 0 will create a new value (and hence a new token) when it reads the value from process N. However, none of the processes ever creates a new value not in the prefix.

(In the case of process 0 the prefix changes so that the statement remains correct.) As $M \geq N$, the value of process 0's variable (by incrementing) eventually reaches a value that is different from all other values in the ring. This value will spread out and all other values will vanish until the prefix is the whole ring. Then we have reached a legitimate state. As no further tokens are ever introduced, the system remains in legitimate states forever.

Conventional proof (outline). [Mer98] formalizes this argument with the following definitions: The system's state space is the $N+1$ fold Cartesian product of $\{0,1,\ldots,M\}$, denoted by $\mathcal{S} = \{0,1,\ldots,M\}^{N+1}$. Component $i+1$ of a tuple in $\mathcal{S}$ denotes the value of the variable of process i. For a system state $s \in \mathcal{S}$, the set $\mathit{prefix}(s) \subseteq \{0,1,\ldots,N\}$ contains the numbers of all those processes which, together with all their neighbors in counter clockwise direction up to process 0, hold the same value as process 0. In the example configuration $\mathit{prefix}(s) = \{0,1,2\}$ and $s = (3,3,3,4,4,7,3,6)$. Set $\mathit{others}(s) \subset \{0,1,\ldots,M\}$ is the set of values held by processes not in the prefix. In our example $\mathit{others}(s) = \{3,4,6,7\}$.[14] At any system state, $\mathit{minfree}(s) \in \{0,1,\ldots,M\}$ is the smallest value by which the value of process 0's variable must be incremented (modulo $M+1$) to get a value that is different from all values in *others*. Hence, *minfree* is the number of times process 0 has to increase its variable until a value is reached which is not present elsewhere in the ring. In the example $\mathit{minfree}(s) = 2$, because 5 is not in *others*. The bit vector $\mathit{enabs}(s) \in \mathbb{B}^N$ is defined such that component i of the vector is true iff process i is enabled. The status of process 0 is not part of *enabs*. In the example $\mathit{enabs}(s) = (f,f,t,f,t,t,t)$, where we write t for true and f for false. Based on these definitions [Mer98] uses valuation function $\mathit{meas}(s) = (\mathit{minfree}(s), \mathit{enabs}(s))$ from system states to the set $\{0,1,\ldots,M\} \times \mathbb{B}^N$ with well-founded order $\sqsubseteq_l$, where $\sqsubseteq_l$ denotes the lexicographic order derived from the usual less-or-equal order on $\{0,1,\ldots,M\}$ and the order $\sqsubseteq_{\mathbb{B}}$ on the booleans given by $f \sqsubseteq_{\mathbb{B}} t$ and $x \sqsubseteq_{\mathbb{B}} x$ for $x \in \mathbb{B}$.[15]

The proof in [Mer98] consists of two basic parts. First, it is shown that once a legitimate state is reached, the system remains in legitimate states forever. Second, [Mer98] shows that the system eventually reaches a state where only one process is enabled. The second part proceeds by proving that steps of processes different from process 0 decrease *enabs* without increasing *minfree*, while steps of process 0 decrease *minfree* as long as a further process exists which is enabled. Hence, it is proven that *meas* strictly decreases with every step of the system which starts in a non-legitimate state. As *meas* is strictly decreasing for steps from these states and as $\sqsubseteq_l$ is well-founded, a finite number of steps must lead to a state where *meas* no longer strictly decreases, i.e. to a legitimate state. Note that when a legitimate state is reached, succeeding steps of the system may increase *meas*.

[14] Value 3 is held by process 6 which is not in the prefix.

[15] For partially ordered sets $(A, \sqsubseteq_A)$ and $(B, \sqsubseteq_B)$, the lexicographic order $\sqsubseteq_{lex}$ on $A \times B$ is defined by $(a,b) \sqsubseteq_{lex} (a',b')$ iff $a \sqsubseteq_A a' \wedge a \neq a'$ or $a = a' \wedge b \sqsubseteq_B b'$. The lexicographic order is well-founded if the orders $\sqsubseteq_A$ and $\sqsubseteq_B$ are well-founded [Win93].

Proof based on attraction. In order to prove the same property of self-stabilization we can employ the specialization of Theorem 6.3 for discrete systems introduced in Section 6.4.2.1 with a slight variant of valuation function *meas* as Liapunov function. The set A, whose attraction we show, consists of all those system states in which exactly one process is enabled. The topology we use on $\mathcal{S}$ is defined by $\mathfrak{O}_{\mathcal{S}} = \{\emptyset, A, \mathcal{S}\}$. As A is an open set in $\mathfrak{O}_{\mathcal{S}}$, proving attraction of A shows that A is entered in finite time and never left again (see Section 6.3.4.2). This exactly is the goal behind self-stabilization. The Liapunov function L we use results from a modification of *meas*. We extend the assessment space by a new bottom element, $V = \{0, 1, \ldots, M\} \times \mathbb{B}^N \cup \{\bot\}$. The lexicographic order $\sqsubseteq_l$ is carried over to V by defining $x \sqsubseteq_V y$ iff either $x = \bot$ or $x \sqsubseteq_l y$. Therefore, $\bot$ is the least element in V and $\sqsubseteq_V$ is a well-founded order on V. Now $L \in \mathcal{S} \to V$ is defined by $L(s) = meas(s)$ iff more than one process is enabled in s and $L(s) = \bot$ otherwise. By definition of L and by the topology on $\mathcal{S}$, $V^+ = V$ holds. Furthermore, L is monotonously decreasing along the traces of the system, because of the respective property of *meas* as explained above and because of the new element $\bot$. Adding a bottom element is necessary, because as soon as the first legitimate state is reached *meas* no longer is non-increasing. By adding the bottom element we simply identify all legitimate states in the assessment space V of the system. Third, as long as $\bot$ is not reached L is strictly decreasing at the time instants where a step of the system is scheduled, because *meas* also is. Finally, $L^{-1}(\bot) = A$ holds by definition of L. Hence, function L satisfies the properties required in the specialization of Theorem 6.3 for discrete systems defined in Section 6.4.2.1 and A therefore is an attractive set. Openness of A in the considered topology yields that attraction implies that A is eventually entered and never left again, regardless of the initial state. Consequently the system is self-stabilizing.

Note that L also satisfies requirement 3 in Theorem 6.1, because for any v we can choose $A \in N(A)$. The other requirements of Theorem 6.1 are satisfied as well, because they correspond to those of Theorem 6.3 and L satisfies a specialization of this theorem. Hence, A is also stable which, together with the openness of A, implies that it is an invariant set, i.e. it is never left by the system. Furthermore, note that we could also have used a finer topology such as the topology $\{S_0, S_1, \ldots, S_{N+1}\}$, where S_i is the set of states with at most i tokens. Hence, $S_0 = \emptyset$, $S_1 = A$ and $S_{N+1} = \mathcal{S}$. With this topology the proof would be the same, because A is open in it as well.

6.6 Discussion and Further Work

6.6.1 Contribution

Based on an abstract system model which is suitable for hybrid systems and close to models in computer science, we have formalized important properties of control systems. The properties have been extracted from the evaluation of nine case studies

and a number of textbooks on control theory. The properties have been classified w.r.t. the semantic models relative to which they are defined. Rating the properties according to the number of case studies in which they were considered identified invariance properties as the most important ones. Furthermore, the rating revealed that a refinement notion based on trace inclusion preserves (at least) those properties which were regarded most frequently in the case studies.

For the properties of stability and attraction the vital role topology plays in their definition was made obvious. Topologies were identified under which stability and attraction are equivalent to invariance and persistence, respectively. The latter two are important classes of properties in computer science. Due to their importance in control theory, proof methods for stability and attraction were examined in greater detail. This study resulted in an adaptation of the general Liapunov-like proof methods from [MT75] to our model of hybrid systems. Furthermore, we were able to formalize parallels between the Liapunov theory and abstraction in computer science. Namely, circumstances were identified under which Liapunov functions define Galois connections.

We applied the developed proof methods to two example systems. First, we considered a small hybrid example system and proved stability and attraction for specific assumptions about the environment and w.r.t. a certain meaningful topology for the system. Elaborating the example revealed that the proof is made more difficult by the imprecision which is present in the model w.r.t. the values (and times, respectively) for which discrete transitions are taken. In fact, considerable time had to be spent to identify circumstances in which stability can be proven and is not violated by the imprecision in the model. Although this is undesirable, it merely reflects that stability properties of an ideal, exact model can not necessarily be transfered to a real model. Apparently, additional assumptions about the environment and/or a more liberal interpretation of the considered properties are necessary to carry over results from a precise to an imprecise model.

As a second example, we demonstrated how the developed methods can be applied to a purely discrete self-stabilizing algorithm, and we compared the resulting proof with the conventional proof. While the proof is not simplified by our methods, it is nevertheless interesting to see how the intuition of bad and good system configurations is reflected in the topology underlying the new proof. This indicates that topology, which in this respect can be seen as identifying equivalence classes, also has relevance for practical computer science systems, not only for theoretical results.

Note that the outlined proof methods for stability and attraction are not meant to replace the variety of existing techniques. Instead, existing techniques can and should be used to construct optimal stable controllers in single control modes. The techniques of the kind presented here are only necessary if stability has to be ensured in the presence of switching between control modes. For instance, in process automation there are many interesting example systems where stability in only one mode is desired,

e.g. the wire stretching plant in [PPS00]. However, the proof methods for stability and attraction developed here could be particularly useful in applications where there are also disturbances in the discrete state space. An example is the work on robotic hand regrasping [SHB+99] where controllers are designed which are supposed to compensate discrete errors resulting from incorrect environment models.

With the continuing integration of software and its physical environment in many systems we conjecture that properties which have only been of interest for continuous systems so far, like stability and attraction, will also become important for the software part of embedded systems. Furthermore, progress in mobile, self-configuring systems and in component-based software engineering may also make stability and attraction important for computer science.

6.6.2 Related Work

Dynamical systems theory. [MT75] studies similar properties from a very general system theory point of view. In particular, [MT75] also introduces an extension of the classical Liapunov theory (see e.g. [Lue79]) for their systems. We build on this result by extending it to nondeterministic systems, which play an important role in the early development phases of discrete (computer science) systems. [MW95] also puts the Liapunov theory into a more general framework and regards it as one way of defining abstractions, i.e. qualitatively equivalent comparison systems to the system under study. This work, however, remains limited to systems operating on metric spaces, because it assumes metric spaces as the basis for the abstraction mappings it defines.

[Sin96] mentions that there is a correspondence between invariance in control theory and safety in computer science and between attraction in control theory and liveness in computer science without going into further detail. Furthermore, Sintzoff notes the similarity between Liapunov functions in control theory and Floyd functions used in computer science to prove termination of programs [Sin92]. With our formal definitions of control systems' properties, we make these correspondences precise by identifying classes of topologies where the correspondences become apparent.

[SG95] and [Geu96] regard dynamical systems from an abstract, systems theory point of view, based on predicate transformers ([SG95]) or iterated relations ([Geu96]). The authors define the concepts of invariance, fullness of invariants (i.e. invariants are not empty) and atomicity of invariants (invariants are singletons) and (finite time) attraction. Furthermore, invariants and attractors are identified as necessary or potential, corresponding to universal or existential path quantification over a system's computations. [Geu96] mentions the correspondence between necessary invariants and invariants in linear-time temporal logic in computer science, and between necessary attractors and termination in computer science. Moreover, the authors elaborate the

relationship between the notion of *chaos* from dynamical systems theory and fullness and atomicity of invariants.

The classification into necessary and potential properties is similar to our partitioning of properties into universal and existential properties. However, for existential properties we furthermore distinguish between the system input and the system output (or state), which is in the spirit of alternating-time temporal logic (ATL) [AHK97]. This is necessary to classify the control theory properties of controllability and observability.

The proof methods for atomicity of invariants and attraction given in [SG95, Geu96] are also Liapunov-like criteria. In Section 6.4.2.1, a criterion from [SG95] was used to derive a specialization of our general proof method for attraction.

In [Geu96], the exposition of the properties is motivated by investigating which kind of dynamics are created by (discrete) dynamical systems. This classification of dynamics serves as the basis for analyzing the effect of composition operators on the dynamics. In contrast, this work focuses towards hybrid systems development. We are mainly interested in properties actually considered for hybrid systems during their design. This explains why we based our exposition of properties to a large part on the evaluation of case studies. Continuous dynamics, which play an important role here, are not considered in [Geu96] and correspondingly topologies on the input, output or state space are not essential for the meaning of properties there.

[Aki93] develops the topological foundations of general dynamical systems starting from iterated relations. Although invariance and attraction (also in the context of Liapunov theory) are considered there, the theory does not seem to be immediately useful for the application to hybrid systems. It rather supports a deeper understanding of dynamical systems in general.

Computer science. The role of topology in computer science, namely for the classification into safety and liveness properties, is explained in [AS85]. Note that the specific topology used in this context is defined on the space of traces, not on the state space, as in our state-based versions of stability and attraction. [Mis96] discusses advantages and disadvantages of using domain theory instead of topology as a basis for computer science.

A number of temporal logics have been defined for hybrid systems, most notably the *hybrid temporal logic* of [HMP93], the extension of the duration calculus in [CRH93], hybrid TLA+ [Lam93] and [Fri98a]. The logics in [HMP93] and [CRH93] are interval temporal logics, and [Fri98a] defines a first order logic suitable for hybrid systems. However, these papers do not study specific properties of hybrid systems such as stability and attraction.

Control theory. [Bra94] defines a method for proving stability of hybrid systems. It basically consists of requiring the existence of a Liapunov function for each individual control mode (or *control state*, in the sense of HyCharts). Furthermore, whenever the

control state is changed, the value of the Liapunov function of the entered control state is required to be less or equal to the value of the Liapunov function of the left control state. This method is strongly focused on proving the stability of continuous variables in the presence of mode switching. It does not allow incorporation of a topology on the control state space. All control states are regarded as equivalent. Hence, it mainly concentrates on the continuous fragment of the dynamics of hybrid systems, discrete jumps in the (otherwise continuous) variables are not considered in the underlying system model and the proposed proof method.[16] Thus, it can be regarded as specialization of this work, which definitely is highly relevant for practice.

6.6.3 Further Work

A first topic of further work is that specializations of the general proof methods developed in this chapter are necessary in order to make them more useful in practice. Such specializations should be driven by deficits encountered and experience gained in the practical development of hybrid systems. Apart from that, it is also necessary to validate within case studies whether existing proof principles from computer science which have been adapted to hybrid systems (e.g. [Pnu94, Lam93]) are suitable in applications.

A second highly relevant topic of further work is the search for methods that allow proving properties of complex hybrid systems by analyzing the behavior of their components. [Geu96] analyzes whether invariance properties and attraction of (sub)systems are preserved under composition and iteration of the relations, which are used to define discrete systems. A result is that hardly any interesting properties are maintained in general under composition operators yielding complex dynamics, such as union of relations and connected products, i.e. products of relations with a kind of interaction. Our definition of systems imposes stronger constraints on them and our disjoint sum/additive visual attachment used for the semantics definition of HySCharts (Sections 3.3.3 and 3.3.4) is a special case of union of relations in [Geu96]. Therefore, there is hope that there is a greater potential for compositional reasoning at least under disjoint sum. In fact, the proof method for stability of switched and hybrid systems presented in [Bra94] can be interpreted as a case in which circumstances are identified where the stability of individual systems is preserved under the disjoint sum.

As far as compositional reasoning for stability proofs under parallel composition/multiplicative visual attachment and feedback (Section 3.3.2) of system components is concerned, the situation is less hopeful. This judgment is motivated by the observation that stability inherently is a global property which assumes that a (sub)system is considered in parallel with its environment, not in isolation.

[16] However, it seems they can be integrated easily.

Chapter 7

Summary and Conclusion

This chapter gives a summary of the thesis and draws some general conclusions. Directions for further work are indicated.

7.1 Summary

There are two basic contributions in the thesis. First, it proposes elements of an infrastructure for the systematic development of hybrid systems. Second, it provides deeper insight into important properties of hybrid systems by formalizing them and relating them to computer science.

The thesis has pointed out that integrated notations and methods for hybrid systems, which consider both the discrete as well as the continuous dynamics of such systems, are highly desirable in order to reduce design risks and gain flexibility. A development process for hybrid systems has been suggested which tries to achieve this aim by postponing the implementation-related partitioning of a hybrid system into discrete-time and continuous-time subsystems to the later development phases. The idea is to validate important system properties already in early phases and to transfer these properties to the fine-grained models occurring in later phases. For the feasibility of this approach, suitable notations and validation methods are needed. Furthermore, model transformations are required which guarantee that previously established properties remain valid. The thesis contributes to these prerequisites by defining appropriate notations and by introducing model transformations for them.

The introduced notations are formal and visual. They allow the unambiguous specification of the data flow in a hybrid system and the control flow in its components. The notations are close to standard techniques used in software engineering, such as UML and Statecharts, and they allow the integration of (simple) block diagrams from control theory. This stresses their adequacy for use in practice. Their semantic foundation is based on an abstract machine model for hybrid computation and on an algebraic

theory of hierarchic graphs. The computation model helps to clearly separate the discrete and continuous dynamics in hybrid systems on the level of semantics. As a side effect of the algebraic theory, the axioms of the algebra can be used to manipulate the visual notations.

For the transformation of systems, known refinement techniques for discrete systems, which preserve vital classes of properties, have been examined w.r.t. their validity for the introduced hybrid notations. The result is that some techniques can rather directly be reduced to axioms of the algebra which is the semantic foundation for the notations.

A central result of the thesis is the identification of conditions under which a specification in discrete time is a (formal) refinement of a specification in an underlying continuous time model. Although this is stated for the notations used in this thesis, the general principle can also be carried over to other hybrid automata-like description techniques for hybrid systems. Methods have been presented which enable the systematic construction of a discrete-time specification from a continuous-time specification such that the identified conditions for refinement hold. These methods involve the integration of techniques from numerical mathematics and control theory in order to obtain a discrete-time version of the analog dynamics. An important point for the refinement is that it is (usually) only possible if the continuous-time specification allows some uncertainty; it may not be arbitrarily precise.

The chosen notion of refinement is motivated by a detailed study of important properties of hybrid systems. The thesis has formally defined such properties and has classified them. Furthermore, for the important concepts of stability and attraction, proof methods have been examined. Here, the thesis puts emphasis on identifying parallels between properties and techniques from control theory and those from computer science. In particular, parallels between stability and invariance, and attraction and persistence have been formalized. Additionally the thesis has been able to substantiate that Liapunov functions, which are a means for proving stability in control theory, in specific cases correspond to Galois connections, which are used for abstraction in computer science. With this detailed examination of properties of hybrid systems, the thesis fosters a better understanding of these systems by computer scientists.

Summary for the running example. All fundamental concepts of the thesis have been demonstrated along the example system *electronic height control* (EHC). In retrospect, the proof methods for the properties stability and attraction were applied to a highly abstract model of the system. This model also shows that the kind of uncertainties which are needed to enable its later discrete-time implementation do not lead to too much nondeterministic behavior, but instead result in rational behavior, which has the desired properties. A more concrete version of the EHC has been used as example for the discretization. This version results from splitting the abstract system into several subcomponents. The split into subcomponents was not considered explicitly in the thesis, but it can have effects on stability and attraction, because it

introduces a delay in the communication between controlling device and controlled device. The discretization of a part of the more concrete model, however, is known to maintain stability and attraction. Thus, the model resulting from the discretization is guaranteed to be stable and attractive, if the split of the abstract model into subcomponents also maintained these properties.

7.2 Conclusion

The simple transferability of some known refinement techniques to HyCharts, i.e. to the two kinds of notations introduced in this thesis, demonstrates the utility of an algebraic framework for systems development. Simple equivalence transformations directly result from the axioms of the algebra. Furthermore, the algebraic framework allows us to define a denotational semantics for key concepts of Statecharts, which are included in HyCharts, in a simple manner. Nevertheless, the graph algebra-based denotational semantics of one of the two notations has also been complemented by an equivalent operational semantics in the thesis (cf. Appendix A.1.2). This operational semantics, to a large extent, also builds upon the graph algebra. It was motivated by some proofs which were easier to conduct with the operational semantics as a basis. Possibly the reason is that thinking in terms of operational steps of a system is often more natural than thinking in terms of fixed points.

As far as the time-discretization methods for component specifications with an underlying continuous time model are concerned, it is important to note that various knowledge of the dynamics of a component's input is required to ensure that the discretization is a (rational) refinement of the original specification. The minimum time between events, Lipschitz constants and possible errors associated with continuous periods of evolution are typically needed. Provided such data is given by the designer, automation of some steps in the construction of the discrete-time component is possible. Although the needed knowledge hampers the immediate application of these methods in practice, the methods make conditions explicit which guarantee that the discretized system behaves "like" the original one. Thus, even if a designer only informally considers the conditions which are required to hold for refinement and only discharges them by means of his or her engineering experience, the situation is greatly improved. Without the explicit conditions, it would remain uncertain which factors have to be observed when a discrete-time implementation for a hybrid component is designed.

A further characteristic of the introduced discretization methods is that a specification with uncertainties in regard to the exact value of variables and in regard to the exact timing is required as a starting point. For a specification which assumes ideal computation, i.e. error-free values and absence of any delays, refinement to a discrete-time implementation is in general impossible. The larger a specification gets, the higher the danger of a propagation of such uncertainties or errors is. From the point of view of

high accuracy, it therefore is desirable to employ discretization before a specification is split into very fine-grained subcomponents. In practice, a trade-off is necessary here because too early discretization, on the other hand, is contrary to the aim of extensively validating an abstract model before decisions are made which may be difficult to alter later on, such as the discretization.

The study of properties of hybrid systems contained in this thesis formally relates properties of control systems and some of their proof methods to known classes of properties in computer science. With the continuing integration of software and its physical environment in many systems, we conjecture that such properties as stability and attraction will also become important to the software part of embedded systems. This stresses the importance of a coherent view on these properties not only for hybrid systems.

As a concluding remark, a study of engineering disciplines such as control theory is in general highly desirable for the aim of making software design more like other kinds of engineering, where the systematic usage of mathematical models at appropriate points in the development process is a matter of course.

7.3 Further Work

With respect to the methodology for hybrid systems development, a closer integration of sequence chart notations should be explored further. These notations can help to narrow the gap to informal, textual requirements. Systems development could greatly benefit from this.

A primary starting point for further work is to examine when properties of an ideal model can be transformed to relaxed properties which hold for a realistic model with uncertainties. However, as pointed out in Section 5.1, this appears to be very difficult even for simple properties.

Tool support is mandatory for the application of the developed discretization techniques in practice. As outlined in Section 5.7, automation of some steps in these techniques is possible. Primary candidate tools for implementing this are *MaSiEd* [SPP01] and also *Charon* [AGH+00], because the notations used there closely correspond to HyCharts.

Besides that, it is advisable to explore specializations of the techniques developed in this thesis for specific subclasses of hybrid systems. For instance, finding discrete-time refinements of hybrid components is simplified if components only communicate via events. Similarly, the general proof methods provided in Section 6.4 can also be tuned for specific classes of systems.

Apart from these topics immediately related to the thesis, further work on validation and verification techniques for hybrid systems is also necessary. Particularly relevant

themes in this field are (hybrid) model checkers with increased scalability and testing techniques which try to exploit properties of the continuous dynamics in hybrid systems.

Appendix A

Proofs

A.1 Proofs about HyCharts and DiCharts

A.1.1 Totality of HySCharts Revisited

In Section 3.2.4 totality of relation *Cmp*, which defines the semantics of the hybrid computation model and therefore of HySCharts, was considered. Totality of *Cmp* means that for any start state $s \in n \cdot \mathcal{S}$ and input $\iota \in \mathcal{I}^{\mathbb{R}_{p+}}$, $Cmp(s)(\iota)$ is nonempty. The proof given in Section 3.2.4 proceeds inductively. It constructs an infinite stream τ as the limit of a sequence of streams which satisfy the properties required by *Cmp* on a sequence of increasing finite intervals. The upper bound of these intervals is required to diverge such that they cover whole $\mathbb{R}_+$. For this proof principle to be valid, we must show that *Cmp* has a certain closure property which ensures that the limit τ of the constructed sequence of streams indeed is in *Cmp*.

Before we can do so, it is important to note that the proof of totality implicitly uses a version of *Cmp* defined for *finite* intervals when it argues about finite streams satisfying the properties required by *Cmp*. We write Cmp_t to denote this finite time semantics of a hybrid machine on time interval $[0, t)$, $t \in \mathbb{R}_+$. Similarly, we write St_t to denote its finite time state-based semantics. The type of Cmp_t is $n \cdot \mathcal{S} \to (\mathcal{I}^{[0,t)} \to \mathcal{P}(\mathcal{O}^{[0,t)}))$ and for St_t it is $n \cdot \mathcal{S} \to (\mathcal{I}^{[0,t)} \to \mathcal{P}((n \cdot \mathcal{S})^{[0,t)}))$, where we write $M^{[0,t)}$ to denote the piecewise smooth, piecewise Lipschitz continuous functions from $[0, t)$ to M.[1] The definitions of Cmp_t and St_t are derived from the relational expressions for *Cmp* and *St*. For *Cmp* this relational expression is given in Example 3.4 of Section 3.3.2. For *St* the expression is similar, but with $Out^\dagger$ replaced by the identity:[2]

$$St(s) \quad = \quad ((\mathbb{R}_2^{\mathcal{I}} \times \mathsf{I}_{n \cdot \mathcal{S}}) \;;_\times\; (\mathsf{I}_{\mathcal{I}} \times Com^\dagger) \;;_\times\; Ana \;;_\times\; \mathbb{R}_2^{n \cdot \mathcal{S}} ;_\times\; (\mathsf{I}_{n \cdot \mathcal{S}} \times Lim_s)) \uparrow_\times^{n \cdot \mathcal{S}}$$

[1] Lipschitz continuity and smoothness are defined as in Section 3.2.3.

[2] *St* therefore allows us to refer to a component's internal state.

We define St_t (and similarly Cmp_t) by interpreting the multiplicative graph operators in this expression w.r.t. time model $[0,t)$. This requires to replace the analog part Ana in the expression by a finite time version Ana_t of it. Ana_t is defined by restricting each activity Act occurring in Ana to $[0,t)$, i.e. $Act|_{[0,t)}$, adapting the definition of discontinuity adaption $da(.)$ (Section 3.2.3) to these finite activities by regarding finite streams and interpreting the time-extended additive graph operators w.r.t. time model $[0,t)$. Moreover, when interpreting the above expression w.r.t. time model $[0,t)$ the time extension of the discrete part Com and the output projection Out, in the case of Cmp_t, also is only performed w.r.t. that time model. Note that the multiplicative and time-extended additive graph operators have already been defined for various kinds of time models in Sections 3.3.2 and 3.3.4. A formula for St_t which results from expanding the definitions of the graph operators occurs in the proof of Theorem A.2.

We now need some preliminary definitions before we can proceed to proving the closure property required from Cmp and St, respectively:

Definition A.1 (Time-divergent prefix monotonous sequence.) *Let (t_i), $t_i \in \mathbb{R}_+$, be an infinite increasing sequence of time points with limit ∞ and let (μ_i) be an infinite sequence of piecewise smooth, piecewise Lipschitz continuous streams such that $\mu_i \in X^{[0,t_i)}$ and each μ_i is a prefix of μ_{i+1}, formally $\mu_i = \mu_{i+1}|_{[0,t_i)}$ for $t_i < \infty$. Then (μ_i) is called a* time-divergent prefix monotonous sequence.

Time-divergent prefix monotonous sequences are similar to prefix monotonous sequences as defined for discrete streams, e.g., in [Kah74, Win93], but their definition only includes sequences of streams for which time diverges. With the natural numbers as time model, the limit of an infinite sequence of strictly prefix monotonous streams automatically is infinite, because every stream in the sequence is at least "one element longer" than its predecessor, i.e. it is defined for at least one more time unit. With the real numbers as time model, however, we can have an infinite sequence of strictly prefix monotonous streams for which time does not diverge. An example is the sequence (ν_i), where each ν_i is constantly 0 and defined on $[0,t_i)$, with $t_i = \Sigma_{k=0}^{i} \frac{1}{2}^k$. For $i \to \infty$, t_i converges to 2. In the context of semantics Cmp, we are only interested in infinite streams. This motivates to explicitly restrict our attention to sequences for which time diverges.

The set of finite and infinite dense streams $\bigcup_{t>0} X^{[0,t)} \cup X^{\mathbb{R}_{p+}}$ is a complete metric space w.r.t. the metric $d_{\overline{s}}$ in Definition B.20 (Appendix B.5). Thus, we can use metric space theory to argue about limits of sequences.

Definition A.2 (Divergence closure.) *Let $M_t \subseteq X^{[0,t)}$ for all $t > 0$ and $M \subseteq X^{\mathbb{R}_{p+}}$ be a family of sets of dense streams. $\bigcup_{t>0} M_t \cup M$ is called* divergence closed *iff the limit μ of any time-divergent prefix monotonous sequence (μ_i) with $\mu_i \in M_{t_i}$ for all i and divergent time sequence (t_i) is in M.*

Note that the limit μ is guaranteed to exist in $X^{\mathbb{R}_{p+}}$, because (μ_i) is a Cauchy sequence w.r.t. the complete metric space of finite and infinite dense streams $(\bigcup_{t>0} X^{[0,t)} \cup X^{\mathbb{R}_{p+}}, d_{\bar{s}})$, and (t_i) diverges (see Appendix B.5). Furthermore, μ is uniquely determined by sequence (μ_i), because we are in a metric space (Appendix B.4). Due to the metric used (Definition B.20), every element μ_i of the sequence is a prefix of μ, i.e. $\mu_i = \mu|_{[0,t_i)}$. This will be used in the following proofs in this section. Divergence closure of $\bigcup_{t>0} M_t \cup M$ is similar to topological closure of $\bigcup_{t>0} M_t \cup M$ w.r.t. the metric space of finite and infinite dense streams, but weaker (see Theorem B.10). In contrast to topological closure of $\bigcup_{t>0} M_t \cup M$ divergence closure only consideres specific converging sequences. It focuses on transferring results from the finite time M_t to the infinite time M, which is what we are interest in here. Hence, sequences and limits of sequences within M are not considered.

Divergence closure is introduced, because it allows us to conclude that the limit of a (time-divergent prefix monotonous) sequence, as constructed in the proof of totality of *Cmp* (Section 3.2.4), indeed is in *Cmp*. Before we come to the divergence closure of HySChart semantics we need a lemma concerning activities.

Lemma A.1 *Let Act be the semantics of a primitive activity or a finite sequential composition of activities. The set* $\bigcup_{t>0} Act|_{[0,t)} \cup Act$ *is divergence closed.*

Proof. Let (α_i) be a time-divergent prefix monotonous sequence with $\alpha_i|_{[0,t_i)} \in Act|_{[0,t_i)}$ and limit α. The proof proceeds by induction over the structure of *Act*. *Act* may result from the semantics of a differential constraint, a stability constraint or the conjunction of such constraints (Sections 3.5.2 and 5.4.3.3). Furthermore, as assumed in this lemma, it may consist of the sequential composition of such semantics.[3]

In the first case *Act* is the semantics of a differential equation or differential constraint $\dot{\vec{x}} \# f(\vec{x}, \vec{i})$, $\# \in \{\geq, \leq, =, >, <, \neq\}$, as defined in Section 3.5.2. By the semantics definition for such constraints, if α in not in *Act*, there must be a point in time $t \in \mathbb{R}_+$ for which the constraint is false. As the elements of (α_i) are prefixes of α (because of the underlying metric space), there is a prefix α_i of α with $t \in [0, t_i)$. Thus, the constraint is already false for α_i which contradicts the assumption. In the case of *Act* being the semantics of a stability constraint as defined in Section 5.4.3 the argument is similar, as such stability constaints also are defined via universal quantification over time.

For *Act* given as the semantics of a (finite) conjunction of constraints $[\![a_1 \wedge a_2]\!]$ note that the semantics of conjunction is defined as the intersection of the semantics of the individual activities. By the induction hypothesis these semantics $[\![a_1]\!]_{[0,t)}, [\![a_1]\!]$ and $[\![a_2]\!]_{[0,t)}, [\![a_2]\!]$ are divergence closed. Hence, for a sequence (α_i) with $\alpha_i|_{[0,t_i)} \in [\![a_1 \wedge a_2]\!]|_{[0,t_i)}$ we have $\alpha \in [\![a_j]\!]$, $j = 1, 2$, which implies $\alpha \in [\![a_1]\!] \cap [\![a_2]\!]$. This is

[3] Note that differential and stability constraints, and conjunction are syntactic entities, while sequential composition is defined on the semantics of activities. With the term activity we usually refer to its semantics.

similar to the preservation of closure in Topology by the finite intersection of closed sets.

Now consider the finite sequential composition of activities. As explained in Section 3.5.2, for any activity $(\iota, \varphi, \psi) \in Act$ implies $\varphi = \psi$. With some set arithmetic, this provides that the sequential composition of activities is equal to their intersection. Thus, this case is similar to conjunction above. □

Theorem A.2 *The set* $\bigcup_{t>0} St_t(s)(\iota|_{[0,t)}) \cup St(s)(\iota)$ *is divergence closed for any initial state* s *and any input stream* ι. *As a shorthand, we will also say that* St_t, St is divergence closed *to denote this property in the sequel.*

Proof. Let (τ_i) be a time-divergent prefix monotonous sequence with limit τ and $\tau_i \in St_{t_i}(s)(\iota|_{[0,t_i)})$ for a divergent time sequence (t_i) and an initial state s.

First, a closer look at St_t is necessary. Unfolding the definitions of the graph operators results in:

$$\begin{aligned} St_t(s) \;=\; \{(\iota|_{[0,t)}), \tau) \in \mathcal{I}^{[0,t)} \times (n \cdot \mathcal{S})^{[0,t)} \mid \exists \sigma \in (n \cdot \mathcal{S})^{[0,t)}. \\ \sigma \in Com^{\dagger}(\iota|_{[0,t)}), Lim_s(\tau)) \;\wedge \quad (1) \\ \tau \in Ana_t(\iota|_{[0,t)}), \sigma)\} \quad (2) \end{aligned}$$

where for convenience we use a relational notation for St_t. The formula for the infinite time version St is similar, but without the time restrictions (see Section 3.2.4). Thus, for each τ_i with $\tau_i \in St_{t_i}(s)(\iota|_{[0,t_i)})$ there is a σ_i for which (1) and (2) hold. As explained in Section 3.5.2, $\psi \in Ana(\iota, \varphi)$ implies $\psi = \varphi$. By expanding the definition of the analog part's finite time semantics Ana_t it is easy to prove that the same holds for Ana_t. Thus, the τ_i uniquely determine finite streams σ_i by $\sigma_i = \tau_i$ for which (1) and (2) hold. Obviously, (σ_i) converges to $\sigma = \tau$.

Now we prove $\tau \in St(s)(\iota)$ by establishing that (the infinite time versions of) conjunct (1) and (2) in the formula for St hold for ι, σ and τ. First, we regard conjunct (1). Unfolding the time extension † and Lim_s in (1) results in a formula which universally quantifies over time t and relates σ, ι and τ at (and just before) any point in time t. Hence, if this subformula is falsified, it is already falsified for finite t which in turn implies $\sigma_i \notin Com^{\dagger}(\iota|_{[0,t_i)}, Lim_s(\tau_i))$ for some i. Thus, $\tau_i \notin St_{t_i}(s)(\iota|_{[0,t_i)})$ which cannot be true because of the assumptions.

In the following we establish that the second conjunct (2) in the formula of St is true for ι, σ and τ. According to Section 3.5.2 Ana can be written as a time-extended disjoint sum of adapted, sequentially composed activities, i.e. $Ana = +_{j=1}^{n} da(Act_j)$ where each Act_j refers to the sequentially composed activities associated with control state j.[4] Unfolding the definitions of $+$ and $da(.)$ yields that $(\kappa, \gamma) \in Ana(\alpha, (\kappa, \beta))$ holds for input stream α, data-state streams β and γ, and control-state stream κ iff on each interval where α and (κ, β) are smooth, $((\alpha, \beta), \gamma)$ restricted to that interval conforms

[4] Remember that discontinuity adaption distributes over sequential composition (Section 3.5.2).

to the currently active sequential composition of activities. More formally the following must hold: $\forall \delta, m.\, \kappa|_\delta = m^\dagger|_\delta \ \wedge \ (\alpha,\beta)|_\delta \in (\mathcal{I}^{\mathbb{R}_{s+}} \times \mathcal{S}^{\mathbb{R}_{s+}})|_\delta \Rightarrow ((\alpha,\beta),\gamma)|_\delta \in Act_m|_\delta$ where δ is a subinterval of $\mathbb{R}_+$. Ana_t is similar, but δ is a subinterval of $[0,t)$ in that case.

Let us write $c\tau$, $d\tau$ and $c\sigma$, $d\sigma$ for the control-state stream and the data-state stream of τ and σ, respectively, i.e. $\tau = (c\tau, d\tau)$ and $\sigma = (c\sigma, d\sigma)$. Furthermore, we use a similar notation for the control-state streams and the data-state streams of the elements of the sequences (τ_i) and (σ_i) in the following. If (2) was false for τ and σ, there must be an interval δ on which ι and σ are smooth, but $d\sigma$ and $d\tau$ do not conform to the activity associated with $c\sigma$ on δ. If δ is a finite interval, it is contained in $[0,t_i)$ for some i and (2) is already violated for τ_i and St_{t_i} which violates the assumptions. Thus, δ must be infinite and not covered by $[0,t_i)$ for any i. By definition of sequences (τ_i) and (σ_i), $\tau_i \in Ana_{t_i}(\iota|_{[0,t_i)}, \sigma_i)$ holds for all i. As ι and σ are smooth on δ, they are also smooth on $\delta \cap [0,t_i) = [t',t_i)$ for some t' and $((\iota, d\sigma_i), d\tau_i)|_{[t',t_i)} \in Act_k|_{[t',t_i)}$ holds for all i with $t_i > t'$ and $k = c\sigma_i(t')$. Using that activities are independent from absolute timing we can shift ι, σ and τ to the left by t'. Then divergence closure of activities (Lemma A.1) provides that the limit of the shifted streams is in the shifted activity. Applying time independence again to shift this back to the right yields the desired result, $((\iota, d\sigma), d\tau)|_{[t',\infty)} \in Act_k|_{[t',\infty)}$.

Hence, $\sigma \in Com^\dagger(\iota, Lim_s(\tau))$ and $\tau \in Ana(\iota,\sigma)$ holds, i.e. $\tau \in St(s)(\iota)$ as required for divergence closure of St_t, St. □

The corresponding property of Cmp, i.e. divergence closure of $\bigcup_{t>0} Cmp_t(s)(\iota|_{[0,t)}) \cup Cmp(s)(\iota)$ for any initial state s and any input stream ι, is a consequence of divergence closure of St_t, St, because Cmp results from St by applying $Out^\dagger$ which is defined in a pointwise manner (Section 3.2.4).

Regarding the proof of totality of Cmp in Section 3.2.4 again, we see that the proof amounts to constructing a time-divergent prefix monotonous sequence (σ_i) with divergent time sequence (t_i) and $\sigma_i \in St_{t_i}$. The above theorem ensures that limit σ is indeed in St and $Out^\dagger(\sigma)$ therefore is in Cmp.

Divergence closure of DiSCharts (Section 5.3) could be proved in a similar way. As we do not need divergence closure of DiSCharts in the thesis, we do not elaborate on this.

A.1.2 Inductive Reasoning for HySCharts

As already indicated in Section 3.2.4 the definition of the analog part, which cuts streams into segments on which Com is idle, the input is smooth and the state stream conforms to an activity, can be regarded as implicitly defining an operational semantics for HySCharts. Namely, we can think of $\sigma \in St(s)(\iota)$ as consisting of a finite of infinite sequence of smooth segments, which conform to an activity, such that Com idles on

each segment and the starting point of each segment is an output of *Com* for the present input and the last point of the previous segment.[5] If σ is a finite sequence of segments the last segment is required to be infinitely long. This corresponds to cases in which the discrete part is only needed for a finite time period after which the behavior remains smooth. In this section we use the idea of inductively pasting such smooth segments to develop an operational way to construct the output streams in the state-based semantics *St*. This gives us an inductive proof principle for properties of HySCharts.

Below, we first show that every prefix of a stream which is an output of *St* also is an output of the finite time version St_t, which was introduced in Appendix A.1.1 (Corollary A.4 below). Then, we establish that every output of St_t can be constructed operationally by pasting smooth segments in the way sketched above (Theorem A.5 below). The prefixes of an output stream σ of *St* can be used to define a time-divergent prefix monotonous sequence of streams σ_i which are outputs of St_t and which converge to σ. As every one of these streams can be constructed operationally due to the previous statement, this yields that every output of *St* is the limit of an operationally constructed sequence of streams. In essence, this means that there is no output of *St* which is not covered by the operational construction of sequences of segments. This yields an inductive proof principle for properties of HySCharts (Theorem A.6 below).[6] These results can be transfered to the black box semantics *Cmp* of HySCharts by output projection.

Prefix closure. We start with a theorem relating the semantics of hierarchic graphs in different time models.

Theorem A.3 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two time models of the kind used for multiplicative hierarchic graphs (see Section 3.3.2.1) such that $\mathcal{T}_1$ ends before $\mathcal{T}_2$ and both consist either of natural or of real numbers. Let R be the relational expression (i.e. the textual representation) for a multiplicative hierarchic graph with primitive nodes $n_1, \ldots, n_k$ (Section 3.3.2), or let it be the expression for a time extended additive hierarchic graph, which only consists of the operators and connectors defined in Section 3.3.4, discontinuity adaption $da(.)$ and the primitive nodes $n_1, \ldots, n_k$. Furthermore, let $r_i \subseteq A_i^{\mathcal{T}_2} \times B_i^{\mathcal{T}_2}$, $i \in \{1, \ldots, k\}$ be relations interpreting the nodes n_i. Interpreting the operators and connectors in R w.r.t. time model $\mathcal{T}_2$ and restricting the resulting relation to the shorter time model $\mathcal{T}_1$ yields a subset of the relation resulting from restricting the r_i to $\mathcal{T}_1$ first and then interpreting R w.r.t. time model $\mathcal{T}_1$. Formally: $(R_{\mathcal{T}_2}(r_1, \ldots, r_k))|_{\mathcal{T}_1} \subseteq R_{\mathcal{T}_1}(r_1|_{\mathcal{T}_1}, \ldots, r_k|_{\mathcal{T}_1})$, where we write $R_{\mathcal{T}_i}$ for the semantics of expression R interpreted w.r.t. time model $\mathcal{T}_i$. $R_{\mathcal{T}_i}$ depends on the interpretations $r_1, \ldots, r_k$ of the primitive nodes $n_1, \ldots, n_k$.*

[5] As before, we write *St* to denote the state-based denotational semantics of the hybrid computation model and therefore of HySCharts. *Cmp* refers to the black box semantics.

[6] Note that Theorem A.2 ensures that we do not regard superfluous streams.

Proof. The proof is by structural induction and follows immediately from the definitions of the operators and connectors. As example, we regard the case of multiplicative feedback $R \uparrow_{\times}^{C}$ for relational expression R and relations $r_i \subseteq A_i^{\mathcal{T}_2} \times B_i^{\mathcal{T}_2}$, $i \in \{1, \ldots, k\}$, which interpret the primitive nodes in the graph corresponding to R. Here, R has type $(A^{\mathcal{T}} \times C^{\mathcal{T}}) \times (B^{\mathcal{T}} \times C^{\mathcal{T}})$ for some sets A, B and C and time model $\mathcal{T}$. By the induction hypothesis $(R_{\mathcal{T}_2}(r_1, \ldots, r_k))|_{\mathcal{T}_1} \subseteq R_{\mathcal{T}_1}(r_1|_{\mathcal{T}_1}, \ldots, r_k|_{\mathcal{T}_1})$ holds. For any $a \in A^{\mathcal{T}_2}$, $b \in B^{\mathcal{T}_2}$, and $c \in C^{\mathcal{T}_2}$, the hypothesis and $(a, c) \in (R_{\mathcal{T}_2}(r_1, \ldots, r_k))(b, c)$ obviously implies that $c|_{\mathcal{T}_1}$ also is a fixed point w.r.t. the shorter time model: $(a, c)|_{\mathcal{T}_1} \in (R_{\mathcal{T}_1}(r_1|_{\mathcal{T}_1}, \ldots, r_k|_{\mathcal{T}_1}))(b, c)|_{\mathcal{T}_1}$. Due to the definition of multiplicative feedback this yields $(R'_{\mathcal{T}_2}(r_1, \ldots, r_k))|_{\mathcal{T}_1} \subseteq R'_{\mathcal{T}_1}(r_1|_{\mathcal{T}_1}, \ldots, r_k|_{\mathcal{T}_1})$, where we write $R'_{\mathcal{T}_i}$ for the interpretation of relational expression $R \uparrow_{\times}^{C}$ w.r.t. time model $\mathcal{T}_i$. □

As a direct result of the theorem we get the following corollary by using the relational expression for St given in Appendix A.1.1.

Corollary A.4 *Let $s \in n \cdot \mathcal{S}$ and $\iota \in \mathcal{I}^{\mathbb{R}_{p+}}$. If $\sigma \in St(s)(\iota)$ then for all $t \in \mathbb{R}_+$ $\sigma|_{[0,t)} \in St_t(s)(\iota|_{[0,t)})$ holds.*

Note that this corollary is similar to *prefix closure* of set $\bigcup_{t>0} St_t(s)(\iota|_{[0,t)}) \cup St(s)(\iota)$ as defined e.g. in [Wec92].

Operational construction. Now transition relations are defined which allow us to construct the (finite) streams produced by St_t. We define the *discrete step* relation $\stackrel{}{\twoheadrightarrow}_i$ by $s \stackrel{}{\twoheadrightarrow}_i s'$ iff $s' \in Com(i, s)$.

The *analog step* relation $\rightsquigarrow_{\iota}^{\tau}$ is defined by $s \rightsquigarrow_{\iota}^{\tau} s'$ iff

- $\iota \in [t_1, t_2) \to \mathcal{I}$ for some $t_1 < t_2$ in $\mathbb{R}_+$,
- $\tau \in [t_1, t_2) \to n \cdot \mathcal{S}$,
- ι and τ are smooth and Lipschitz continuous,
- τ initially is s, $\tau(t_1) = s$,
- the "last" value of τ is s', $\lim_{x \nearrow t_2} \tau(x) = s'$,
- and τ satisfies Ana such that Com is idle, formally $(\iota, \tau, \tau) \in Ana|_{[t_1, t_2)}$ and for all $t \in [t_1, t_2)$, $\tau(t) \in Com(\iota(t), \tau(t))$.[7]

Thus, during a time interval covered by an analog step the input is smooth and Lipschitz continuous and Com is idle.

The discrete step and analog step relations are very similar to the time- and transition step relations which define the operational semantics of hybrid automata [ACH+95]. The sequential composition of the discrete step relation with the analog step relation is called the *hybrid step* relation. We can regard St_t as inductively built by the finite iteration of hybrid steps:

[7] Note that $\tau(t) = \lim_{x \nearrow t} \tau(x)$ for $t \in (t_1, t_2)$ since τ is continuous.

Theorem A.5 *The set of streams in $St_t(s)(\iota|_{[0,t)})$ for any $t > 0$ and the set of streams generated by the finite iteration of hybrid steps starting from $s \in n \cdot \mathcal{S}$ for input $\iota \in \mathcal{I}^{\mathbb{R}_{p+}}$ are identical.*

Proof. Let $s \in n \cdot \mathcal{S}$ and $\iota \in \mathcal{I}^{\mathbb{R}_{p+}}$. First, note that for $\iota|_{[0,t)}$ and any stream $\sigma \in St_t(s)(\iota|_{[0,t)})$ the interval $[0,t)$ can be partitioned into finitely many subintervals such that the segments of ι and σ which result from restricting them to such a subinterval both are smooth and Lipschitz continuous on the respective subinterval. This holds, because $\mathcal{I}^{\mathbb{R}_{p+}}$ and St_t only contains piecewise smooth, piecewise Lipschitz continuous trajectories and interval $[0,t)$ is finite. A stream generated by the finite iteration of hybrid steps naturally consists of finitely many such smooth, Lipschitz continuous segments. With induction we prove that a stream consisting of finitely many smooth, Lipschitz continuous segments is in St_t iff it is generated by the finite iteration of hybrid steps. Let $[0,t_1)$ be the first interval where ι and σ are smooth and Lipschitz continuous. The first segment $\tau \in (n \cdot \mathcal{S})^{[0,t_1)}$ of σ is in $St_{t_1}(s)(\iota|_{[0,t_1)})$ iff it results from a hybrid step from s, i.e. $s \multimap_{\iota(0)} ; \rightsquigarrow^{\tau}_{\iota'} \lim_{x \nearrow t_1} \tau(x)$ where ; is the sequential composition of relations and $\iota' = \iota|_{[0,t_1)}$. This is easy to show if we regard the defining formula for St_t where the graph operators are unfolded, as in the proof of Theorem A.2. Assume that for the first i intervals where ι and σ are both smooth and Lipschitz continuous, $\sigma|_{[0,t_i)}$ is in $St_{t_i}(s)(\iota|_{[0,t_i)})$ and it is also generated by the i-fold iteration of hybrid steps (induction hypothesis). Then, for the $(i+1)$st interval where ι and σ are smooth and Lipschitz continuous, $\sigma|_{[0,t_{i+1})}$ is in $St_{t_{i+1}}(s)(\iota|_{[0,t_{i+1})})$ iff $\sigma|_{[t_i,t_{i+1})}$ results from a hybrid step from $\lim_{x \nearrow t_i} \sigma(x)$, i.e. $\lim_{x \nearrow t_i} \sigma(x) \multimap_{\iota(t_i)} ; \rightsquigarrow^{\tau}_{\iota'} \lim_{x \nearrow t_{i+1}} \tau(x)$ for $\tau = \sigma|_{[t_i,t_{i+1})}$ and $\iota' = \iota|_{[t_i,t_{i+1})}$. Again this immediately follows from unfolding the definitions of the graph operators in St_t. □

Trace induction. Based on these results we can define induction over the streams (or traces) of a HySChart as a proof principle for properties of a HySChart. We assume no specific notation for properties, but assume that a property P is given as a set of hybrid input and state streams, $P \subseteq \mathcal{I}^{\mathbb{R}_{p+}} \times (n \cdot \mathcal{S})^{\mathbb{R}_{p+}}$. For instance, some kind of temporal logic or HySCs [GKS00] can be used to define such a set.

Theorem A.6 (Trace induction.) *Let $St(s)$ be the semantics of a HySChart with initial state s and let P be a property such that $\bigcup_{t>0} P|_{[0,t)} \cup P$ is divergence closed. To establish that P holds for $St(s)$, i.e. $St(s) \subseteq P$ (where $St(s)$ is regarded as a set), it suffices to prove the following for all input streams $\iota \in \mathcal{I}^{\mathbb{R}_{p+}}$:*

1. *For all $\tau \in \mathcal{S}^{[0,\delta)}$ resulting from a single hybrid step from s for input $\iota|_{[0,\delta)}$ show that $(\iota|_{[0,\delta)}, \tau) \in P|_{[0,\delta)}$ holds.*
2. *Assuming $St_t(s) \subseteq P|_{[0,t)}$ prove that for all extensions $\sigma' \in \mathcal{S}^{[0,t')}$, $t' > t$, of any $\sigma \in St_t(s)(\iota|_{[0,t)})$ by a hybrid step for input $\iota|_{[t,t')}$, $(\iota|_{[0,t')}, \sigma') \in P|_{[0,t')}$ holds. As a stronger obligation, the claim may alternatively be proven for all hybrid step extensions σ' of any $(\iota|_{[0,t)}, \sigma) \in P|_{[0,t)}$ for input $\iota|_{[t,t')}$.*

Proof. Due to Theorem A.5 the hybrid step relation may be used to generate all elements in $St_t(s)(\iota|_{[0,t)})$ for any input ι. Thus, $St_t(s) \subseteq P|_{[0,t)}$ for all $t > 0$ follows if claims 1 and 2 are established. (Note that to result in a legal component $St(s)$ must produce infinite output upon any input.)

Now we prove $St(s) \subseteq P$. Any $\sigma \in St(s)(\iota)$ can be used to define a time-divergent prefix monotonous sequence (σ_i) for divergent time sequence (t_i) such that $\sigma_i|_{[0,t_i)} \in St_{t_i}(s)(\iota|_{[0,t_i)})$ (Corollary A.4). Similarly, a further time-divergent prefix monotonous sequence (ι_i) results from setting $\iota_i = \iota|_{[0,t_i)}$. By the argument above $(\iota|_{[0,t_i)}, \sigma_i|_{[0,t_i)})$ is also in $P|_{[0,t_i)}$. Together with divergence closure of $\bigcup_{t>0} P|_{[0,t)} \cup P$ this yields $(\iota, \sigma) \in P$. □

Divergence closure of $\bigcup_{t>0} P|_{[0,t)} \cup P$ corresponds to *admissibility* of property P in the context of usual fixed point induction. Informally, admissibility requires that if a property holds for each element of a monotonous sequence, it also holds for the limit of the sequence [Pau87, Win93]. Liveness properties, for instance, are not admissible (and also not divergence closed) w.r.t. prefix monotonous sequences. As St_t, St is divergence closed (Theorem A.2), $St(s)$ and, hence HySCharts, do not formalize liveness properties. This is usual for conventional automata models without any fairness assumptions. (Nevertheless a HySChart can satisfy a liveness property. For instance, the liveness property of eventually producing a certain output y after receiving input x is satisfied by a HySChart which always produces y within two time units after receiving x.)

If P is defined by time extension, i.e. $P = p^\dagger$ for $p \subseteq \mathcal{I} \times \mathcal{S}$ and † denotes time extension (see Section 3.2.4), trace induction corresponds to computational induction as defined in [Pnu94]. In such a case p corresponds to the so called *state formulas* in temporal logic [MP92] and divergence closure automatically holds (see below).

Lemma A.7 *For $p \subseteq \mathcal{I} \times \mathcal{S}$ divergence closure of $\bigcup_{t>0} (p^\dagger)|_{[0,t)} \cup p^\dagger$ holds.*

Proof. Let μ be the limit of a time divergent prefix monotonous sequence (μ_i) with $\mu_i \in p^\dagger|_{[0,t_i)}$ for divergent time sequence (t_i). Assume μ is not in $p^\dagger$. Then, by definition, there must be a time instant t with $\mu(t) \notin p$. Due to the underlying metric space, every μ_i is a prefix of μ (cf. Appendix A.1.1). Therefore, this implies that one of the μ_i must already have failed to satisfy p which contradicts the assumption. □

We use trace induction in the example of Section 6.5.1, where the proof principle can even be simplified further as all considered discrete steps do not affect the quantities considered there.

In the sense of Section 6.3.5 we only regarded universal properties in Theorem A.6, i.e. every possible behavior in St must satisfy the property. For divergence closed properties demanding the *existence* of certain streams in St, induction based on the hybrid step relation may also be used, because St_t, St is divergence closed.

A.1.3 Time Guardedness of HySCharts

In this section we establish that a component whose behavior is defined by a HySChart is time guarded. To do so we first prove that the finite time semantics of HySCharts coincides with the time restriction of the infinite time semantics. Again we use the state based semantics St and its finite time version St_t, $t > 0$ (Section A.1.1).

Theorem A.8 *For any start state s and input $\iota \in \mathcal{I}^{\mathbb{R}_{p+}}$, the finite time semantics of a HySChart coincides with the time restriction of its infinite time semantics. Formally, $St_t(s)(\iota|_{[0,t)}) = (St(s)(\iota))|_{[0,t)}$ for all $t > 0$.*

Proof. Direction "$\supseteq$" is a consequence of Corollary A.4.

The other direction relies on totality of HySCharts and uses that the future behavior of a HySChart at any point in time is entirely determined by its state at that time and the (future) input. In other words, a HySChart has no further knowledge about its past besides the information encoded in its state. This is utilized by employing the hybrid step semantics of HySCharts. The proof principle is as follows: We consider the final state of a state stream φ in the finite time semantics St_t. Then, a sequence of extensions of φ is constructed by using this state as an initial state for an input stream which is shifted in time. The way the construction is performed (via the hybrid step relation) ensures that the infinite stream which is the limit of the constructed sequence is in the infinite time semantics St. As φ is a prefix of that infinite stream in St, time restriction of St completes the proof. The technical details are presented below.

Let φ be in the finite time semantics $St_t(s)(\iota|_{[0,t)})$ for a $t > 0$, and let s' be the "final state" of φ, i.e. $s' = \lim_{x \nearrow t} \varphi(x)$. Furthermore, we define $\iota' \in \mathcal{I}^{\mathbb{R}_{p+}}$ as the left shift of ι by t, $\iota' = \iota^{-t}|_{\mathbb{R}_+}$ where $\tau^{-t}(x)$ is defined as $\tau(x + t)$ for any stream τ. As we require that HySCharts define total components, $St(s')(\iota')$ is nonempty and we can select a $\sigma' \in (n \cdot \mathcal{S})^{\mathbb{R}_{p+}}$ from this set. We use the prefixes of σ' and ι' to define two time divergent prefix monotonous sequences[8] (σ'_i) and (ι'_i) for a divergent time sequence (t_i) with $\sigma'_i \in St_{t_i}(s')(\iota'_i)$ for all i. The limits of these sequences are σ' and ι', respectively. By Theorem A.5 each σ'_i is generated by the finite iteration of hybrid steps starting from s' for input ι'_i. Similarly, φ also results from the finite iteration of hybrid steps starting from s for input $\iota|_{[0,t)}$. The hybrid step relation is independent from absolute time, because all activities occurring in *Ana* are (Section 3.2.3), and *Com* is defined in a pointwise manner. Hence, we can shift each σ'_i and ι'_i to the right by t and paste the two finite sequences of hybrid steps generating φ and σ'_i at state s'. For each σ'_i, this yields a stream σ_i which is generated by the finite iteration of hybrid steps starting from state s for input ι_i, where $\sigma_i|_{[0,t)} = \varphi$, $\sigma_i|_{[t,t+t_i)} = \sigma_i'^t$ (right shift of σ'_i by t, see above), and similarly $\iota_i = \iota|_{[0,t+t_i)}$.

[8] see Definition A.1

Theorem A.5 yields that $\sigma_i \in St_{t+t_i}(s)(\iota|_{[0,t+t_i)})$ for all i. Moreover, sequence (σ_i) is a time divergent prefix monotonous sequence, because (σ_i') is. Divergence closure of St_x, St (Theorem A.2) provides that the limit σ of this sequence is in $\sigma \in St(s)(\iota)$. Thus, $\varphi \in St(s)(\iota)|_{[0,t)}$. □

Theorem A.9 *HySCharts define time guarded components.*

Proof. Let St be the state based semantics of a HySChart. Let s be a start state and let ι_1 and ι_2 be input streams with coincident prefix up to time t, $\iota_1|_{[0,t)} = \iota_2|_{[0,t)}$. We prove $St(s)(\iota_1)|_{[0,t)} = St(s)(\iota_2)|_{[0,t)}$.

Let $\sigma \in St(s)(\iota_1)|_{[0,t)}$. The application of Theorem A.8 and the assumption yields $\sigma \in St_t(s)(\iota_2|_{[0,t)})$. Using Theorem A.8 again, we get $\sigma \in St(s)(\iota_2)|_{[0,t)}$. The other direction follows from symmetry.

For the "black box" semantics *Cmp* which results from adding output projection *Out* to St, its time guardedness is an immediate consequence. □

A.2 Discretization as Refinement

A.2.1 Operational Semantics of DiSCharts

Similar to Appendix A.1.2 we define transition relations for DiSCharts (cf. Section 5.3) here. The relations are needed for the proof of the refinement principle given in Section 5.4.

As the transition relations only regard finite streams, we first show that all finite prefixes of an infinite stream produced by a DiSChart also satisfy the finite time version of DiSChart semantics. As for HySCharts, we argue about DiSChart semantics without output projection $Out^\dagger$ in order to be able to consider all state information. We write DSt for the semantics of DiSChart without the projection. The finite time version of DSt, denoted by DSt_k for finite time models $\mathcal{T}_k = \{0, \ldots, k\}$, $k \in \mathbb{N}$, is obtained by interpreting the multiplicative graph operators in the defining equation of DSt w.r.t. this time model (see Section 3.3.2):

$$\begin{array}{lcl} DSt_k & \in & n \cdot \mathcal{S} \to (\mathcal{I}^{\mathcal{T}_k} \to \mathcal{P}((n \cdot \mathcal{S})^{\mathcal{T}_k})) \\ DSt_k(s) & = & ((\wp_2^{\mathcal{I}} \times \mathsf{I}_{n \cdot \mathcal{S}}) \;;_\times\; (\mathsf{I}_{\mathcal{I}} \times DAna_k) \;;_\times\; DCom^\dagger \;;_\times\; \wp_2^{n \cdot \mathcal{S}} ;_\times\; (\mathsf{I}_{n \cdot \mathcal{S}} \times \Delta_s)) \uparrow_\times^{n \cdot \mathcal{S}} \end{array}$$

where $DAna_k$ is defined by restricting each activity $DAct$ occurring in the expression for the infinite time $DAna$ to $\mathcal{T}_k$, i.e. $DAct|_{\mathcal{T}_k}$, adapting the definition of discontinuity adaption $da(.)$ (Section 5.3.3) to these finite activities by regarding finite streams and interpreting the time-extended additive graph operators w.r.t. time model $\mathcal{T}_k$ (Section 3.3.4).

Theorem A.10 *Let $s \in n \cdot \mathcal{S}$ and $\iota \in \mathcal{I}^{\mathbb{N}}$. If $\sigma \in DSt(s)(\iota)$, then for all $k \in \mathbb{N}$, $\sigma|_{\mathcal{T}_k} \in St_k(s)(\iota|_{\mathcal{T}_k})$ holds, where $\mathcal{T}_k = \{0, \ldots, k\}$.*

Proof. The theorem is an immediate consequence of Theorem A.3 which regards discrete as well as continuous time models. □

Discrete-time versions of the discrete, continuous and hybrid step relations (cf. Appendix A.1.2) are defined as follows. In order to avoid ambiguous terminology we call these relations logic step, control step and discrete-time hybrid step relation, respectively. The *logic step* (or discrete-time discrete step) relation $\oplus\!\!\Rightarrow_i$ is given by $s \oplus\!\!\Rightarrow_i s'$ iff $s' \in DCom(i, s)$ and $s'.rs = true$. The additional restriction concerning rs is needed for compatibility with the control step relation defined in the following. The logic step relation expresses the effect of discrete moves by $DCom$ in the machine model for DiSCharts (cf. Figure 5.6).

The *control step* (or discrete-time analog step) relation $\rightsquigarrow_\iota^\tau$ is defined by $s \rightsquigarrow_\iota^\tau s'$ iff

- $\iota \in \{k_1 + 1, \ldots, k_2 + 1\} \to \mathcal{I}$ for some $k_1 \leq k_2$ in $\mathbb{N}$,
- $\tau' \in \{k_1, \ldots, k_2 + 1\} \to n \cdot \mathcal{S}$ and $\tau = \tau'|_{\{k_1, \ldots, k_2\}}$,
- $\tau'(k_1) = s$,
- $s.rs = true$ and $\tau'.rs(k) = false$ for $k \in \{k_1 + 1, \ldots, k_2 + 1\}$,
- $\tau'(k_2 + 1) = s'$,
- τ' satisfies $DAna$ such that $DCom$ is idle, formally $(\iota, \tau'^1|_\delta, \tau'|_\delta) \in DAct|_\delta$, where $\delta = \{k_1 + 1, \ldots, k_2 + 1\}$ and τ'^1 denotes the right shift $\tau'^1(x) = \tau'(x - 1)$, and $\tau'(k) \in Com(\iota(k), \tau'(k))$ holds for all $k \in \{k_1 + 1, \ldots, k_2\}$.

In terms of the machine model of DiSCharts in Figure 5.6, τ' shifted to the right is the input of $DAna$ and τ' is its output. The right shift by one time unit is caused by the delay Δ in the machine model. $DCom$ need not be idle for s'. In fact, it must not be idle if we want to continue with a logic step. The control step relation expresses the effect of $DAna$ whenever $DCom$ is idle in the machine model for DiSCharts.

In contrast to their continuous-time counterparts in Section A.1.2 the smoothness restrictions in the analog step relation are replaced by restrictions on the evolution of rs here, since smooth evolution is not a meaningful notion for discrete-time streams. This results in a difference between control steps and analog steps. A control step spans between two non idle steps of $DCom$ and does not consider input discontinuities. In contrast, an analog step can also be ended by a (higher-order) discontinuity in the input. The sequential composition of the logic step relation with the control step relation is called the *discrete-time hybrid step* relation.

We can regard the finite time semantics DSt_k as inductively built by the finite iteration of hybrid steps, starting with a state s' proposed by the initial activation of $DAna$ for start state s with $s.rs = init$. Remember that by definition of $DAna$ this ensures that

$s'.rs = true$ (Section 5.3.3).[9] This is formalized in the following theorem.

Theorem A.11 *Let $\iota \in \mathcal{I}^{\mathbb{N}}$ and $s \in \mathcal{S}$ an initial state, $s.rs = init$. The set of streams in $DSt_k(s)(\iota|_{\{0,\ldots,k\}})$ for any $k \in \mathbb{N}$ and the set of streams generated by the finite iteration of hybrid steps starting from an s' with $(\iota(0), s, s') \in DAna|_{\{0\}}$ for input ι are identical.*

Proof. First, we point out that any state stream $\sigma \in DSt_k(s)(\iota|_{[0,t)})$ can be partitioned into finitely many segments σ_j, $j \in \{1, \ldots, \ell\}$, $\ell \in \mathbb{N}$, such that $DCom$ is idle on each segment and performs a move at the end of each segment. More formally, variable rs, which signals when $DCom$ is not idle and is used for initialization, is true in the first state of every segment σ_j and false for the rest of the segment. This holds, because the discontinuity adaption $da(.)$, which occurs in $DAna$ (cf. Section 5.3.3), cuts σ into such pieces σ_j. A stream generated by the finite iteration of discrete-time hybrid steps by definition consists of finitely many such idle segments.

With induction we now prove that a stream consisting of finitely many idle segments is in DSt_k iff it is generated by the finite iteration of discrete-time hybrid steps. For a stream σ its first idle segment $\sigma_1 = \sigma|_{\{0,\ldots,k_1\}}$, $k_1 \leq k$, is in $DSt_{k_1}(s)(\iota|_{\{0,\ldots,k_1\}})$ iff it results from a discrete-time hybrid step from an s' with $(\iota(0), s, s') \in DAna|_{\{0\}}$, i.e. $s' \Rightarrow_{\iota(0)} ; \rightsquigarrow^{\sigma_1}_{\iota'} u$, where ; is the sequential composition of relations and $\iota' = \iota|_{\{1,\ldots,k_1+1\}}$.[10] To show this we regard the defining formula for DSt_k where the graph operators are unfolded, as in the proof of Theorem A.2 for the continuous-time case:

$$\begin{aligned} DSt_k(s) \;=\; \{(\iota,\tau) \in \mathcal{I}^{\mathcal{T}_k} \times (n\cdot\mathcal{S})^{\mathcal{T}_k} \;\mid\; & \exists \sigma \in (n\cdot\mathcal{S})^{\mathcal{T}_k}. \\ & \sigma \in DAna_k(\iota, \Delta_s(\tau)) \;\wedge \\ & \tau \in DCom^{\dagger}(\iota,\sigma)\} \end{aligned}$$

where $\mathcal{T}_k = \{0, \ldots, k\}$. Like in the continuous-time case, $DAna_k$ is defined by the disjoint sum of activities, adapted to moves of $DCom$ by $da(.)$. The definition of $da(.)$ (Section 5.3.3) provides that $DCom$ at time 0 receives a s' with $(\iota(0), s, s') \in DAna_k|_{\{0\}}$. By definition of $DAna$ and discrete-time activities (Section 5.3.3), $s.rs = init$ guarantees $s'.rs = true$. Now the equivalence follows from the definitions of the step relations.

Assume that the first m idle segments of σ are in $DSt_{k_m}(s)(\iota|_{\{0,\ldots,k_m\}})$, $k_m < k$, and are also generated by the m-fold iteration of discrete-time hybrid steps. Let u be the final state of the m-th hybrid step. Then the first $m+1$ idle segments of σ are in $DSt_{k_{m+1}}(s)(\iota|_{\{0,\ldots,k_{m+1}\}})$, $k_{m+1} \leq k$, iff $\sigma|_{\{k_m,\ldots,k_{m+1}\}}$ results from a hybrid step originating from u, i.e. $u \Rightarrow_{\iota(k_m)} ; \rightsquigarrow^{\tau}_{\iota'} u'$ for $\tau = \sigma|_{\{k_m,\ldots,k_{m+1}\}}$ and $\iota' = \iota|_{\{k_m+1,\ldots,k_{m+1}+1\}}$. Again this immediately follows from unfolding the definitions of the graph operators in DSt_k and the definitions of the step relations. □

[9] As explained in Section 5.3.3, a separate treatment of the first activation of $DAna$ is needed to maintain correct start values for the variables modified by $DAna$.

[10] (Intermediate) state u is not visible outside, but the start value for $DCom$ in the next iteration. Input $\iota(k_1+1)$ is only needed to compute u, not to compute σ_1.

A.2.2 Proofs about Discretization as Refinement

Based on the logic and control step relations defined for DiSCharts in Section A.2.1 we can give the proof of the discrete-time refinement theorem of HySCharts (Theorem 5.1). Like in the theorem we use the following notation. We write $DCmp$ for the semantics of a given discrete-time component specified with a DiSChart, $DCmp \in n \cdot \mathcal{S}_D \to (\mathcal{I}^{\mathbb{N}} \to \mathcal{P}(\mathcal{O}^{\mathbb{N}}))$. We write $DCom$ and $DAna$ to refer to its (discrete-time) discrete and analog part, and $DOut^\dagger$ for its time extended output projection. Similarly, we write Cmp for the semantics of a given continuous-time component specified with a relaxed HySChart, $Cmp \in n \cdot \mathcal{S}_C \to (\mathcal{I}^{\mathbb{R}_{p+}} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_{p+}}))$. Its discrete part, its analog part and its time extended output projection are denoted by $RCom$, $RAna$ and $ROut^\dagger$, respectively, and R_{int} denotes its output relaxation. Furthermore, we write I and S for the sets of considered inputs and evolutions of the state stream. The projection of a $u \in n \cdot \mathcal{S}_D$ on $n \cdot \mathcal{S}_C$ is denoted by $\pi_{n \cdot \mathcal{S}_C}(u)$ and similar for (sets of) streams.

We start with a lemma essentially stating that the discrete-time hybrid step relation refines the continuous-time hybrid step relation. These relations provide the operational semantics for DiSCharts and HySCharts, respectively. They are defined in Sections A.2.1 and A.1.2.

Lemma A.12 *Let assumptions 1, 2, 3, 4 and 5 of Theorem 5.1 hold. For all* $\iota \in I|_{[k_1T,(k_2+1)T]}$, $k_1 \leq k_2$ *in* $\mathbb{N}$, $\iota' = sample_T(\iota)$, *and* $s \in n \cdot \mathcal{S}_D$ *with* $s.now = k_1T$, *the discrete-time hybrid step* $s \dashrightarrow_{\iota'(k_1)} ; \rightsquigarrow^{\tau'}_{\iota'|_{\{k_1+1,\ldots k_2+1\}}} s'$ *for* $\tau' \in (n \cdot \mathcal{S}_D)^{\{k_1,\ldots k_2\}}$ *implies that there is a* $\tau \in (n \cdot \mathcal{S}_C)^{[k_1T,(k_2+1)T)}$ *which results from finitely many continuous-time hybrid steps for input* $\iota|_{[k_1T,(k_2+1)T)}$ *and start state* $\pi_{n \cdot \mathcal{S}_C}(s)$ *where RCom idles at the discrete steps between the analog steps. At sampling times* τ *and* τ' *agree, the "final states" agree, and the hold extension of* τ' *projected on the output space is in the relaxation* R_{int} *of the output projection of* τ *on interval* $[k_1 \cdot T, (k_2+1) \cdot T)$, *formally:*

$$\begin{array}{ll} \forall k \in \{k_1, \ldots, k_2\}. \pi_{n \cdot \mathcal{S}_C}(\tau'(k)) = \tau(kT) \ \wedge & \textit{(sampling times)} \\ \pi_{n \cdot \mathcal{S}_C}(s') = \lim_{x \nearrow (k_2+1)T} \tau(x) \ \wedge & \textit{(“last states”)} \\ hold_{\mathcal{O},T}(DOut^\dagger(\tau')) \subseteq R_{int}(ROut^\dagger(\tau)) & \textit{(hold extension)} \end{array}$$

Furthermore, in s' *the value of the private clock now is correct, i.e.* $s'.now = (k_2+1)T$.

Proof. Let $\iota \in I|_{[k_1T,(k_2+1)T]}$, $k_1 \leq k_2$ in $\mathbb{N}$, $\iota' = sample_T(\iota)$. For the logic step $s \dashrightarrow_{\iota'(k_1)} s''$, $s, s'' \in n \cdot S_D$ with $s.now = k_1T$, the claim $\pi_{n \cdot \mathcal{S}_C}(s) \multimap_{\iota(k_1T)} \pi_{n \cdot \mathcal{S}_C}(s'')$ is a direct consequence of assumption 2 in Theorem 5.1.

By definition of logic steps $s''.rs = true$ and $DCom$ must be able to idle for s'', as s'' is an output of $DCom$. (This holds because of the way a discrete part is constructed from a hierarchic graph, cf. Section 3.5.1.6.) Next we consider the control step $s'' \rightsquigarrow^{\tau'}_{\iota'|_{\{k_1+1,\ldots k_2+1\}}} s'$ for $\tau' \in (n \cdot \mathcal{S}_D)^{\{k_1,\ldots k_2\}}$. The definition of control steps yields

that τ' evolves according to *DAna* and that *DCom* idles for τ' and input ι'. Hence, $\tau'.rs$ is only *true* initially and *false* thereafter. By assumption 4 of Theorem 5.1 this implies the existence of a continuous $\tau \in (n \cdot \mathcal{S}_C)^{[k_1 \cdot T, (k_2+1) \cdot T)}$ which satisfies *RAna*, i.e. $(\iota|_{[k_1 \cdot T, (k_2+1) \cdot T)}, \tau, \tau) \in RAna|_{[k_1 \cdot T, (k_2+1) \cdot T)}$, which is equal to τ' at sampling times, whose "last value" is $\pi_{n \cdot \mathcal{S}_C}(s')$ and for which the *hold* extension of τ' projected on $\mathcal{O}$ is in $R_{int}(ROut^{\dagger}(\tau))$ on $[k_1 \cdot T, (k_2+1) \cdot T)$. Furthermore, as the value of *now* is correct initially, i.e. $\tau.now(k_1 T) = k_1 T$, and *RAna* ensures that it evolves correctly, $\tau.now$ at each moment contains the current time and the "last value" of *now* is $(k_2+1)T$. This consistency property of *now* is needed in the rest of the proof to be able to apply assumption 3 of Theorem 5.1.

Now we construct continuous-time hybrid steps that generate τ. The problem here is to establish that *Com* can idle. As ι and τ are dense streams, they are piecewise smooth and piecewise Lipschitz continuous on left-closed and right-open intervals. Furthermore, they are restricted to a finite interval. Hence (by definition of piecewise smoothness and piecewise Lipschitz continuity), this interval can be partitioned into finitely many subintervals such that ι and τ are smooth and Lipschitz continuous on each of them. We refine this partitioning by furthermore splitting every subinterval at sampling times kT. The resulting partitioning is used for τ and we write τ_j and ι_j, $j \in \{1, \ldots, \ell\}$, $\ell \in \mathbb{N}$, to denote the ℓ smooth, Lipschitz continuous segments of τ and ι, respectively. By induction we prove that an analog step spans each such interval and that a discrete step, where *RCom* idles, connects adjacent intervals:[11]

Streams ι_1 and τ_1 are smooth and Lipschitz continuous on the first interval and the selection of τ (based on assumption 4 of Theorem 5.1) guarantees that they satisfy *RAna*. As *DCom* idles for the start state s'' of τ_1 and as all dynamics allowed by the activities, and therefore also τ_1, are in S (assumption 5 of Theorem 5.1), assumption 3 yields that *RCom* can idle throughout the interval. Hence an analog step $\pi_{n \cdot \mathcal{S}_C}(s'') \rightsquigarrow_{\iota_1}^{\tau_1} u$ to some state u is possible. Let us now consider u. If the right end point of the considered interval is a sampling instant, we know that τ and τ' agree here (assumption 4), and that *DCom* idles provided that this is not the end point of the last interval ℓ. Thus, *RCom* can also idle (assumption 2), a discrete step $u \xrightarrow{\Rightarrow}_i u$ is possible for the current input i at that point, and we can continue the proof with regarding the next interval. If the right end point of the considered interval already is the end of the last interval ℓ, nothing remains to be done. If the right end point of the considered interval is *not* a sampling instant, we nevertheless know that at the last sampling instant in $[k_1 T, (k_2+1)T]$ immediately preceding the regarded end point *DCom* was idle. In the case of the first interval, this sampling instant is $k_1 T$. Using assumption 3 this yields that *RCom* must be able to idle for u and the current input i at that end point, i.e. the step $u \xrightarrow{\Rightarrow}_i u$ is possible. (Assumption 3 furthermore guarantees that *RCom* can idle throughout the next interval, which is needed in the induction step.) This ends the base case for the first subinterval of the partitioning.

[11] Idle steps of *RCom* suffice, since τ is continuous on $[k_1 \cdot T, (k_2+1) \cdot T)$ (assumption 4, see above).

The induction step is similar.

As a result, the control step is split into finitely many analog steps which are pasted together with discrete steps. Due to the selection of the resulting hybrid stream τ (based on assumption 4, see above), τ satisfies the further claims of the lemma, correspondence of τ and τ' at sampling points, correspondence of their projections on $\mathcal{O}$ up to R_{int} between sampling points and correspondence of the "last states". Moreover, the value of now is also correct for the last state s' (see above). □

Now we come to the proof for Theorem 5.1 of Section 5.4.1. We use the same terminology and symbols as in that theorem.

Proof. (Theorem 5.1.) Under the assumptions of Theorem 5.1 we have to show that for given DiSChart semantics $DCmp$, relaxed HySChart semantics Cmp, inputs in I, evolution constraints S, start state $s \in n \cdot \mathcal{S}_D$ with $s.rs = init$, $s.now = 0$ and sampling period T, it holds that for all $\iota \in I$, every stream in the $hold$ extension of $DCmp$ also is in Cmp, formally $\sigma \in (sample_{\mathcal{I},T};_\times DCmp(s);_\times hold_{\mathcal{O},T})(\iota) \Rightarrow \pi_{n\cdot\mathcal{S}_C}(\sigma) \in Cmp(\pi_{n\cdot\mathcal{S}_C}(s))(\iota)$. In the following we argue about the state-based semantics of $DCmp$ and Cmp. As usual, they are denoted by DSt and St, respectively.

For a $\iota \in I$, let σ be in $(sample_{\mathcal{I},T};_\times DSt(s);_\times DOut^\dagger;_\times hold_{\mathcal{O},T})(\iota)$.[12] Furthermore, let σ' be the corresponding discrete-time stream, without the output projection $DOut^\dagger$ and the time extension by $hold_{\mathcal{O},T}$, and let ι' be the sampled version of ι. Hence, $\sigma' \in DSt(s)(\iota')$. We argue about the finite prefixes of σ' which are generated by the finite iteration of discrete time hybrid steps. (Theorem A.11 guarantees that all prefixes of σ' are indeed generated by the finite iteration of such steps.) We show that for every finite prefix $\sigma'|_{\{0,\ldots,k_j\}}$, $k_j \in \mathbb{N}$, of σ' which is generated by j discrete time hybrid steps, there is a dense stream $\tau_j \in (n \cdot \mathcal{S}_C)^{[0,(k_j+1)T)}$ which is in the finite time version of St, $\tau_j \in St_{(k_j+1)T}(\pi_{n\cdot\mathcal{S}_C}(s))(\iota|_{[0,(k_j+1)T)})$, and τ_j projected on the output space and relaxed by R_{int} contains $\sigma|_{[0,(k_j+1)T)}$. Furthermore, τ_j and $\sigma'|_{\{0,\ldots,k_j\}}$ are equal at sampling instants. By definition of the τ_j as given below, they are a time-divergent prefix monotonous sequence for divergent time sequence (t_j) with $t_j = (k_j + 1)T$. Hence, sequence (τ_j) has a limit, denoted by τ (Appendix A.1.1). Due to divergence closure of St_t, St (Theorem A.2) limit τ is in $St(\pi_{n\cdot\mathcal{S}_C}(s))(\iota)$. This finally yields that σ is in $R_{int}(ROut^\dagger(\tau))$, because divergence closure of $ROut^\dagger$ and R_{int} holds by an argument similar to Lemma A.7. By the definition of relaxed HySCharts the claim of the theorem is an immediate consequence of this result.

At this point it remains to define the sequence (τ_j). Constructing τ_j from $\sigma'|_{\{0,\ldots,k_j\}}$ proceeds by induction over the hybrid step relations by which the elements of St_t and DSt_k are built (Theorems A.5 and A.11).

Let $\omega' \in \{0,\ldots,k_1\} \to n \cdot \mathcal{S}$ be the first segment of σ' resulting from the discrete-time hybrid step $u \Rightarrow_{\iota(0)} s' \overset{\omega'}{\underset{\iota|_{\{1,\ldots,k_1+1\}}}{\rightsquigarrow}} s''$ for a u as given by the initialization of

[12] By definition of DSt, $DCmp(s) = DSt(s);_\times DOut^\dagger$ holds. The corresponding equality is valid in the continuous time case for Cmp and St.

$DAna$, i.e. $(\iota(0), s, u) \in DAna|_{\{0\}}$. Lemma A.12 then yields that there exists a corresponding continuous-time hybrid step $\pi_{n\cdot\mathcal{S}_C}(u) \twoheadrightarrow_{\iota(0)} \pi_{n\cdot\mathcal{S}_C}(s') \rightsquigarrow^{\omega}_{\iota|_{[0,(k_1+1)T)}} \pi_{n\cdot\mathcal{S}_C}(s'')$, with $s''.now = (k_1+1)T$, such that ω and ω' agree at sampling times and the *hold* extension of the output projection of ω' is within relaxation R_{int} of the output projection of ω. By definition of the initialization (Section 5.3.3) $\pi_{n\cdot\mathcal{S}_C}(u) = \pi_{n\cdot\mathcal{S}_C}(s)$ which together with Theorem A.5 provides $\omega \in St_{(k_1+1)T}(\pi_{n\cdot\mathcal{S}_C}(s))(\iota|_{[0,(k_1+1)T)})$. We therefore define $\tau_1 = \omega$.

For the induction step, let $\omega' \in DSt_{k_j}(s)(\iota'|_{\{0,\ldots,k_j\}})$ result from the first j discrete-time hybrid steps and let u with $u.now = (k_j+1)T$ be the final state of the last hybrid step. Furthermore, let $\tau_j \in St_{(k_j+1)T}(s)(\iota|_{[0,\ldots,(k_j+1)T)})$ be the corresponding hybrid stream with $\lim_{x\nearrow(k_j+1)T} \tau_j(x) = \pi_{n\cdot\mathcal{S}_C}(u)$. Similar to the base case, we can again apply Lemma A.12 to show that for the next discrete-time hybrid step starting from u there is a corresponding continuous-time hybrid step starting from $\pi_{n\cdot\mathcal{S}_C}(u)$ such that the resulting streams agree at sampling instants and at the last value, and the *hold* extension of the output projection of the discrete-time stream is in the relaxation by R_{int} of the output projected hybrid stream. Extending τ_j with the constructed hybrid stream yields τ_{j+1} which is in $St_{(k_{j+1}+1)T}(\pi_{n\cdot\mathcal{S}_C}(s))(\iota|_{[0,(k_{j+1}+1)T)})$ by Theorem A.5. □

Next we prove Theorem 5.2 of Section 5.4.3.1 which states that (under certain conditions on the independence of the variables which activities modify) a sample-and-hold-refinement of a relaxed analog part can be constructed from the sample-and-hold-refinements of its individual activities. The proof uses the terminology and symbols introduced in Section 5.4.3.1. In particular, the reader is refered to that section for the definitions of the controlled and unconstrained spaces $\mathcal{D}$, $\mathcal{A}$, $\overline{\mathcal{D}}$, and $\overline{\mathcal{A}}$ of discrete-time and continuous-time activities.

Proof. (Theorem 5.2.) The proof proceeds by induction over the structure of the analog part and is rather technical. It uses that activities are the identity on those controlled variables not occurring in them.

We start with the base case where there only is a single activity, i.e. $DAna = da(DAct)$ and $RAna = da(RAct)$. Let τ' satisfy $da(DAct)$ for input ι' on an interval where there is only one reset at the beginning. More formally, let $\iota \in I$, $\iota' \in sample_T(\iota)$ and $\tau' \in (1 \cdot \mathcal{S}_D)^{\{k_1,\ldots,k_2+1\}}$, $k_1 \leq k_2$ in $\mathbb{N}$, with $\tau'.rs(k_1) = true$ and $\tau'.rs(k) = false$ for $k \in \{k_1+1, \ldots, k_2+1\}$, and let $(\iota'|_\delta, \tau'^1|_\delta, \tau'|_\delta) \in DAna|_\delta$, where $\delta = \{k_1+1, \ldots, k_2+1\}$. The control state in τ' is constant, since there is only one primitive control state ($DAna$ does not consist of a disjoint sum, the state space is $1 \cdot \mathcal{S}_D = \mathcal{S}_D$). By the definition of $da(.)$ as cutting the discrete-time state stream into intervals which start with a reset (cf. Section 5.3.3), this ensures that $(\iota'|_\delta, \tau'^1|_\delta, \tau'|_\delta)$ also is in $DAct|_\delta$. With distinguishing between the variables occurring in the definition of $DAct$ and those not occurring in it, we get that the projection of τ' on the controlled space $\mathcal{D}$ of $DAct$ is in $pDAct|_\delta$ for input ι' and that its projection on $\overline{\mathcal{D}}$ satisfies the time-extended additive identity relation $\mathrm{i}_{\overline{\mathcal{D}}}$, i.e. $(\iota'|_\delta, \pi_{\overline{\mathcal{D}}}(\tau'^1)|_\delta, \pi_{\overline{\mathcal{D}}}(\tau')|_\delta) \in \mathrm{i}_{\overline{\mathcal{D}}}|_\delta$. This implies that the projection of τ' on $\overline{\mathcal{D}}$ is a constant function. Using the theorem's assumption provides that there

is a continuous $\tau \in \mathcal{S}_C^{[k_1 \cdot T,(k_2+1)\cdot T)}$ whose projection on the controlled space $\mathcal{A}$ of $RAct$ satisfies $da(pRAct)$ for input ι and which furthermore has the properties required in the theorem w.r.t. τ' projected on $\mathcal{A}$. This means that the two projections agree at sampling times and in their "last value", and the *hold* extension of τ', projected on the output space of $RAct$, is in the relaxation of τ projected on the same space. We choose τ such that its projection on the unconstrained space $\overline{\mathcal{A}}$ of $RAct$ is a constant function with value $\pi_{\overline{\mathcal{A}}}(\tau'(k_1))$. As $RAct$ is the identity relation on $\overline{\mathcal{A}}$, τ satisfies $da(RAct)$ for input ι, i.e. $(\iota|_{[k_1 \cdot T,(k_2+1)\cdot T)}, \tau, \tau) \in da(RAct)|_{[k_1 \cdot T,(k_2+1)\cdot T)}$. Furthermore, it also satisfies that τ and $\pi_{\mathcal{S}_C}(\tau')$ are equal at sampling instants and the *hold* extension of τ' projected on output space $\mathcal{O}$ is in the relaxation of τ. Thus, assumption 4 of Theorem 5.1 holds for $DAna$ and $RAna$.

Assuming that the claim holds for the sequential composition of m activities we next show that it also holds for $m+1$ sequentially composed activities: $DAna = da(DAct_{m+1})\,;_+(;_{+\,j=1}^{m} da(DAct_j))$ and $RAna = da(RAct_{m+1})\,;_+(;_{+\,j=1}^{m} da(RAct_j))$. Let τ' satisfy $DAna$ for input ι' on an interval where there is only one reset at the beginning, as formally defined above. By the definitions of $da(DAct_{m+1})$ and $;_+$ this implies that there is a discrete-time stream $\omega' \in {\mathcal{S}_D}^{\delta}$, for $\delta = \{k_1+1, \ldots, k_2+1\}$ as above, connecting the two sequentially composed parts, i.e. $(\iota'|_\delta, \tau'^1|_\delta, \omega') \in DAct_{m+1}|_\delta$ and $(\iota'|_\delta, \omega', \tau'|_\delta) \in {;_{+\,j=1}^{m}} da(DAct_j)|_\delta$.

Now we use that $DAct$ and $;_{+\,j=1}^{m} da(DAct_j)$ are completely independent in order to be able to apply the assumptions and the induction hypothesis. As $DAct_{m+1}$ and $;_{+\,j=1}^{m} da(DAct_j)$ reference disjoint sets of controlled variables, $DAct_{m+1}$ is the identity on the controlled variables occurring in the $DAct_j$, $j \in \{1, \ldots, m\}$, and vice versa. $DAct_{m+1}$ and the $DAct_j$ all are the identity on the controlled variables occurring in none of them. Hence, the projections of ω' and $\tau'|_\delta$ on the controlled space $\mathcal{D}_{m+1}$ of $DAct_{m+1}$ coincide (except for the value of variable rs) and similarly on the space $\mathcal{D}_{1\ldots m}$ of controlled variables occurring in one of the $DAct_j$, $\pi_{\mathcal{D}_{1\ldots m}}(\tau'^1|_\delta) = \pi_{\mathcal{D}_{1\ldots m}}(\omega')$ holds. Variable rs is set to *false* by the last activity in the sequential composition of the $DAct_j$. This exactly corresponds to the definition of $pDAct_{m+1}$ where the assignment $rs' = false$ is added to the activity in order to consider the activity in isolation (cf. Section 5.4.3.1). This allows us to apply the assumptions and the induction hypothesis.

Applying the assumption to $pDAct_{m+1}$ for $\pi_{\mathcal{D}_{m+1}}(\tau')$ yields that there is a continuous $\omega \in \mathcal{A}_{m+1}^{[k_1 \cdot T,(k_2+1)\cdot T)}$ which satisfies $da(pRAct_{m+1})$ and the properties required in the theorem w.r.t. τ' projected on the controlled space $\mathcal{A}_{m+1}$ of $RAct_{m+1}$. Using the induction hypothesis for $;_{+\,j=1}^{m} da(RAct_j)$ we get that there is a continuous $\sigma \in \mathcal{A}_{1\ldots m}^{[k_1 \cdot T,(k_2+1)\cdot T)}$ which satisfies $;_{+\,j=1}^{m} da(RAct_j)$ and the properties required in the theorem w.r.t. τ' projected on the space $\mathcal{A}_{1\ldots m}$ of controlled variables occurring in one of the $RAct_j$, $j \in \{1, \ldots, m\}$. We define the continuous function $\tau \in {\mathcal{S}_C}^{[k_1 \cdot T,(k_2+1)\cdot T)}$ by $\pi_{\mathcal{A}_{m+1}}(\tau) = \omega$, $\pi_{\mathcal{A}_{1\ldots m}}(\tau) = \sigma$ and $\pi_{\overline{\mathcal{A}}}(\tau)$ is constant and equal to $\pi_{\overline{\mathcal{A}}}(\tau'(k_1))$, where $\overline{\mathcal{A}}$ denotes the space of those variables not occurring in any

activity of the sequential composition. Like in the base case above, τ' and σ' are constant on this space. As activities do not constrain the evolution of variables not occurring in them, we get that τ satisfies $da(pRAct)$ and $;_{+j=1}^{m} da(RAct_j)$ for input ι, i.e. $(\iota|_{[k_1 \cdot T,(k_2+1)\cdot T)}, \tau, \tau) \in da(RAct_{m+1})|_{[k_1 \cdot T,(k_2+1)\cdot T)}$ and $(\iota|_{[k_1 \cdot T,(k_2+1)\cdot T)}, \tau, \tau) \in ;_{+j=1}^{m} da(RAct_j)|_{[k_1 \cdot T,(k_2+1)\cdot T)}$. Furthermore, it also satisfies that τ and $\pi_{\mathcal{S}_C}(\tau')$ are equal at sampling instants and in their "last values", and that the *hold* extension of τ' projected on the output space is in the relaxation of τ projected on the output space. Thus, assumption 4 of Theorem 5.1 holds for $DAna$ and $RAna$ in the case of finite sequential composition.

Finally, we regard analog parts consisting of the n-fold disjoint sum of activities. The base case for $n = 1$ is covered by the sequential composition considered above. It therefore remains to consider the induction step for the $n + 1$ fold sum. Let $DAna = ;_{+j=1}^{m_{n+1}} da(DAct_{n+1,j}) + (+_{i=1}^{n} (;_{+j=1}^{m_i} da(DAct_{i,j})))$ and similarly $RAna = ;_{+j=1}^{m_{n+1}} da(RAct_{n+1,j}) + (+_{i=1}^{n} (;_{+j=1}^{m_i} da(RAct_{i,j})))$. Let τ' satisfy $DAna$ for input ι' on an interval where there is only one reset at the beginning, as formally defined above. The way the reset variable *rs* is used in DiSCharts ensures that the control state is constant during such an interval (Section 5.3.3). Thus, the control state in τ' is constant to some $\ell \in \{1, \ldots, n+1\}$. By the definition of the disjoint sum this implies that τ' satisfies summand ℓ for input ι' throughout the considered interval $\delta = \{k_1+1, \ldots, k_2+1\}$, i.e. $(\iota'|_\delta, \tau'^1|_\delta, \sigma'|_\delta) \in ;_{+j=1}^{m_\ell} da(DAct_{\ell,j})|_\delta$. On each such interval, this reduces the proof obligation for the sum of activities to the respective proof obligation for the sequential composition of activities. We already covered this above. □

A.3 Proofs about Stability and Attraction

Theorem A.13 *In topologies where C and E are open, (C, E) is stable iff $Sys(C) \subseteq E$.*

Proof. "⇒": With choosing $\alpha = E$ stability yields $\forall b \in \beta.\, Sys(b) \subseteq \alpha$ for a $\beta \in N(C)$. By the definition of neighborhood $C \subseteq \beta$ holds. Hence, we also get that all causes in C produce effects in α, $\forall b \in C.\, Sys(b) \subseteq \alpha$.

"⇐": For every neighborhood of E we can choose $\beta = C$. □

Theorem A.14 *In topologies where set A is open, the (state-based) stability of A is equivalent to invariance of A.*

Proof. The proof essentially is a specialization of that for Theorem A.13. □

Theorem A.15 *In topologies where A is open global attraction is equivalent to the property that A is reached in finite time and not left again.*

Proof. "⇒": As A is open, we can select $A \in N(A)$ as neighborhood in the definition of global attraction. This immediately proves the claim.

"⇐": Due to the definition of neighborhood, $A \subseteq \alpha$ holds for any neighborhood of A. As there is a t such that A is reached and not left again after t, this also holds for any of A's neighborhoods. □

Lemma A.16 *The functions abs and conc as defined in Section 6.4.1.1 are well-defined.*

Proof. We start with function *abs*. To see that the supremum in the definition of $abs(\alpha)$ exists in V for $\alpha \in N(A)$ first note that V is complete. Second, the regarded set $\{v \in V^+ \mid \alpha \supseteq L^{-1}(\langle v \rangle_o)\}$, whose supremum is sought, is nonempty. To prove this we can construct an element of this set: As the topology on $\mathcal{S}$ is the coarsest which makes L continuous, Theorem B.3 implies that every open set α can be written as $\alpha = \bigcup_{j \in J} L^{-1}(I_j)$ for an index set J and open intervals I_j. As $\alpha \supseteq A$, there exists a I_j with $\bot \in I_j$. Hence, I_j can be written as $\langle v \rangle_o$ for some $v \neq \bot$. This implies that $\alpha \supseteq L^{-1}(\langle v \rangle_o)$ holds for v, i.e. the set whose supremum is needed is nonempty, and it furthermore guarantees that the supremum is not equal to $\bot$. Thus, $abs(\alpha) \in V^+$.

For function *conc* we have that set $conc(v)$ is open for $v \in V^+$, because L is continuous. Furthermore, $A \subseteq conc(v)$ because $\forall s \in A.\, L(s) = \bot$. Hence, $conc(v) \in N(A)$ for $v \in V^+$. □

Theorem A.17 *The functions abs and conc as defined in Section 6.4.1.1 are a Galois connection between $(N(A), \supseteq)$ and $(V^+, \sqsupseteq)$, i.e. $(N(A), \supseteq)$ and $(V^+, \sqsupseteq)$ are partially ordered sets, abs and conc are monotonous, and $\alpha \supseteq conc(abs(\alpha))$ (extensivity) and $abs(conc(v)) \sqsupseteq v$ (reductivity) holds. In fact even $abs(conc(v)) = v$ is valid.*

Proof. $(N(A), \supseteq)$ and $(V^+, \sqsupseteq)$ obviously are partially ordered sets (V^+ even is densely ordered by assumption). The abstraction function is monotonous: $\alpha \supseteq \beta \Rightarrow abs(\alpha) \sqsupseteq abs(\beta)$, because for $v = abs(\beta)$, $\alpha \supseteq \beta \supseteq L^{-1}(\langle v \rangle_o)$ holds, which implies that the supremum in the definition of $abs(\alpha)$ is at least v.

The concretization function is monotonous: $v_1 \sqsupseteq v_2 \Rightarrow conc(v_1) \supseteq conc(v_2)$ holds, because of the monotonicity of L^{-1} and $\langle v_1 \rangle_o \supseteq \langle v_2 \rangle_o$.

Extensitivity, i.e. $\alpha \supseteq conc(abs(\alpha))$, holds: For $v = abs(\alpha)$ the definition of supremum together with $\sqsupseteq$ being total and *abs* being monotonous implies that $\forall v' \in \langle v \rangle_o.\, \alpha \supseteq L^{-1}(\langle v' \rangle_o)$ is valid. Hence, α also is a superset of the union of these inverse images, $\alpha \supseteq \bigcup_{v' \in \langle v \rangle_o} L^{-1}(\langle v' \rangle_o)$. The union can be rewritten as $L^{-1}(\{v'' \mid \exists v'.\, v \sqsupset v' \sqsupset v''\})$. As V is densely ordered by $\sqsupset$, there is a v' between any two elements of V (Theorem B.1). Therefore, $\{v'' \mid \exists v'.\, v \sqsupset v' \sqsupset v''\} = \{v'' \mid v \sqsupset v''\} = \langle v \rangle_o$ holds, which finally yields that $\alpha \supseteq L^{-1}(\langle v \rangle_o) = conc(v)$.

$abs(conc(v)) = v$, which implies reductivity, holds because L is onto: For $\alpha = conc(v)$ and $v' = v$ surely $\alpha = L^{-1}(\langle v \rangle_o) \supseteq L^{-1}(\langle v' \rangle_o)$ holds. Hence, $abs(\alpha) \sqsupseteq v'$ must hold.

Due to L being onto, there can be no v'' with $v'' \sqsupset v'$ and $\alpha \supseteq L^{-1}(\langle v'' \rangle_o)$. In more detail this is valid, because for $v'' \sqsupset v'$ there is a $\bar{v}$, $v'' \sqsupset \bar{v} \sqsupset v'$, such that, due to L being a function and onto, $L^{-1}(\bar{v}) \cap L^{-1}(\langle v' \rangle_o) = \emptyset$ holds. This in turn implies that $L^{-1}(\langle v'' \rangle_o) \supsetneq L^{-1}(\langle v' \rangle_o)$. Thus, $v' = v$ is the supremum required in the definition of *abs*. □

Theorem A.18 *Given the abstraction and concretization functions abs and conc as defined in Section 6.4.1.1 monotonicity of L along the system traces of Sys_S implies the stability of A w.r.t. the topology considered on $\mathcal{S}$.*

Proof. Let $\alpha \in N(A)$. For $v = abs(\alpha)$ monotonicity of L along the traces of Sys_S implies that $\forall s \in \mathcal{S}. L(s) \sqsubset v \Rightarrow \forall \sigma \in Sys_S(s, \iota). \forall t. L(\sigma(t)) \sqsubseteq L(s)$ for all $\iota \in I$ since $\sigma(0) = s$ for $\sigma \in Sys_S(s, \iota)$. Hence, selecting $\beta = conc(v)$ yields that $\forall s \in \beta. \forall \sigma \in Sys_S(s, \iota). \forall t. L(\sigma(t)) \sqsubseteq L(s) \sqsubset v$. By the definition of *conc*, $L(\sigma(t)) \sqsubset v$ implies $\sigma(t) \in conc(v)$. Together with $\alpha \supseteq conc(abs(\alpha))$ (extensivity) this implies $\sigma(t) \in \alpha$ for all t, $\sigma \in Sys_S(s, \iota)$, $s \in \beta$ and $\iota \in I$. □

Theorem A.19 *Function L, as defined in Section 6.4.1.2, satisfies the requirements of Theorem 6.1.*

Proof. Requirements one and two trivially hold. To verify the third requirement, let $v \in V^+$. We can then choose $\alpha = L^{-1}(\langle v \rangle)$ which is open because of the continuity of L and which contains A because $L(A) = \bot \in \langle v \rangle$. As L-values of elements of α are less or equal v w.r.t. $\sqsubseteq$, the third requirement holds. For the fourth requirement we choose $L(\alpha)$ for $\alpha \in N(A)$ where L is extended to sets by pointwise extension. As L^{-1} is continuous, $L(\alpha)$ is open in V and can therefore be written as the union of a family of sets in the base of the left topology of V, i.e. $L(\alpha) = \bigcup_{v \in J} \langle v \rangle$ where $J \subseteq V^+ = V$. We select a $v \in J$ and get $L^{-1}(\langle v \rangle) \subseteq \alpha$ as demanded by the fourth requirement. □

Appendix B

Mathematical Foundations

B.1 Sets and Orders

Definition B.1 (Order.) *Let X be a set and $\sqsubseteq$ a relation on X, i.e. $\sqsubseteq$ must be a subset of $X \times X$. $\sqsubseteq$ is an* order *iff it is transitive, reflexive and antisymmetric. If for all $x, y \in X$ either $(x, y) \in \sqsubseteq$ or $(y, x) \in \sqsubseteq$ holds, the order is* total. *Otherwise it is* partial. *Usually infix notation is used for orders, i.e. we write $x \sqsubseteq y$ for $(x, y) \in \sqsubseteq$. $(X, \sqsubseteq)$ is called a* partially ordered set, *iff $\sqsubseteq$ is a partial order on X. It is called a* totally ordered set, *iff $\sqsubseteq$ is a total order on X. If X has a least element w.r.t. $\sqsubseteq$ this element is called* bottom element *and denoted by $\bot$.*

Definition B.2 (Complete partial order (Cpo).) *A partially ordered set $(X, \sqsubseteq)$ is called a* complete partial order *iff if there is a least upper bound for any increasing chain $x_1 \sqsubseteq x_2 \sqsubseteq \ldots$ in X.*

Definition B.3 (Well-founded order.) *Let $(X, \sqsubseteq)$ be a partially ordered set. $\sqsubseteq$ is called a* well-founded order *on X iff there is no infinite (strictly) descending sequence $x_1 \sqsupset x_2 \sqsupset x_3 \sqsupset \ldots$ in X, where $x \sqsupset x'$ denotes $x' \sqsubseteq x \ \wedge \ x' \neq x$.*

For an order $\sqsubseteq$ on X we write $\sqsubset$ to denote the irreflexive relation $\sqsubseteq \setminus id_X$, where id_X is the identity relation on X.

Definition B.4 (Densely ordered set [Eng89].) *Let $(X, \sqsubseteq)$ be a totally ordered set. Let $A, B \subseteq X$ be a partitioning of X, i.e. $A \cup B = X$, such that $A, B \neq \emptyset$ and $x \in A$ and $y \in B$ implies $x \sqsubset y$. X is called* densely ordered *by $\sqsubset$ if for any such partitioning either A has no largest element or B has no smallest element.*

Theorem B.1 *Let X be a set densely ordered by $\sqsubset$. Then for all $x, z \in X$ with $x \sqsubset z$ there is a $y \in X$ with $x \sqsubset y \sqsubset z$.*

Proof. The proof proceeds by contraposition. Assume there is no y between x and z. Let $A = \{a \in X \mid a \sqsubseteq x\}$ and $B = \{b \in X \mid x \sqsubset b\}$. Clearly, $A \cup B = X$ and $a \in A \wedge b \in B \Rightarrow a \sqsubset b$ holds. Furthermore, A contains a largest element, x. Due to the assumption, B contains z as its smallest element. This contradicts the assumption that X is a densely ordered set. □

B.2 Some Topology

This section introduces some basic notions from topology. Most of the definitions are taken or adapted from [Eng89].

Definition B.5 (Topological space [Eng89].) *For a set X and a set of subsets of X, denoted by $\mathfrak{O}$, $(X, \mathfrak{O})$ is a* topological space *iff*

- $\emptyset \in \mathfrak{O}$ *and* $X \in \mathfrak{O}$,
- *for* $U_1 \in \mathfrak{O}$ *and* $U_2 \in \mathfrak{O}$, $U_1 \cap U_2 \in \mathfrak{O}$,
- *for* $\mathcal{A} \subseteq \mathfrak{O}$, $\bigcup \mathcal{A} \in \mathfrak{O}$.

X is also called a *space*, $\mathfrak{O}$ is called a *topology* on X, and the elements of $\mathfrak{O}$ are called *open* sets w.r.t. the topology. A set $U \subseteq X$ is called *closed* if its complement $X \setminus U$ is an open set.

For two topologies $\mathfrak{O}_1$ and $\mathfrak{O}_2$ on a set X topology $\mathfrak{O}_1$ is *coarser* than topology $\mathfrak{O}_2$ iff $\mathfrak{O}_1 \subseteq \mathfrak{O}_2$.

Definition B.6 (Neighborhood of a point.) *For $x \in X$, a set $U \in \mathfrak{O}$ is a* neighborhood *of x iff $x \in U$ [Eng89]. We define $N(x)$ to denote the set of all neighborhoods of x, formally $N(x) = \{O \in \mathfrak{O} \mid x \in O\}$.*

The notion of neighborhood is extended so sets in a pointwise manner.

Definition B.7 (Neighborhood of a set.) *$U \in \mathfrak{O}$ is a* neighborhood *of $Y \subseteq X$ iff U is a neighborhood for every element of Y, formally $Y \subseteq U$. The set of all neighborhoods of Y is defined as $N(Y) = \{O \in \mathfrak{O} \mid Y \subseteq O\}$.*

Note that this notion of neighborhood implies that every open set is a neighborhood of itself: $Y \in \mathfrak{O} \Rightarrow Y \in N(Y)$

Example B.1 (Discrete topology.) $(X, \wp(X))$ is a topological space, where $\wp(X)$ is the set of all subsets of X. $\mathfrak{O} = \wp(X)$ is called the *discrete topology* on X [Eng89]. In the discrete topology every set $U \subseteq X$ is open *and* closed.

Definition B.8 (Base for a topological space.) *A set $\mathcal{B} \subseteq \mathfrak{O}$ of open sets is a* base for the topological space $(X, \mathfrak{O})$ *iff every non-empty open set can be represented by a union of sets in $\mathcal{B}$.*

[Eng89] states that any base has the properties:

(B1) for any $B_1, B_2 \in \mathcal{B}$ and for all $x \in B_1 \cap B_2$ there is a $B \in \mathcal{B}$ such that $x \in B \subseteq B_1 \cap B_2$

(B2) for any $x \in X$ there is a $B \in \mathcal{B}$ such that $x \in B$

Theorem B.2 (Topology generated by a base.) *Let $\mathcal{B}$ be a set of subsets of X with the properties (B1) and (B2). Then the set $\mathfrak{O} \subseteq \wp(X)$ consisting of all subsets of X which are unions of sets in $\mathcal{B}$ is a topology on X. Moreover, $\mathcal{B}$ is a base for this topology. $\mathfrak{O}$ is called the* topology generated by base *$\mathcal{B}$.*

Proof. A proof is given in [Eng89]. □

Example B.2 (Left topology.) Let X be a set partially ordered by $\sqsubseteq$. The sets $\langle x \rangle = \{x' \in X \mid x' \sqsubseteq x\}$ for every $x \in X$ generate a topology on X. This topology is called the *left topology on X induced by $\sqsubseteq$* [Eng89].

To show that the proposed sets indeed generate a topology we apply Theorem B.2, which amounts to showing that the sets satisfy the assumptions of the theorem. For the first assumption holds as $x \in \langle x_1 \rangle \cap \langle x_2 \rangle$ implies that $\langle x \rangle \subseteq \langle x_1 \rangle \cap \langle x_2 \rangle$ because of transitivity of $\sqsubseteq$. The second assumption is valid because $x \in \langle x \rangle$ for all $x \in X$.

Example B.3 (Interval topology.) Let X be a set containing at least two elements and totally ordered by $\sqsubseteq$ (i.e. for all $x, y \in X$, $x \sqsubseteq y$ or $y \sqsubseteq x$ holds). $\sqsubseteq$ is also called a *linear order*. We write $x \sqsubset y$ for $x \sqsubseteq y \ \wedge \ x \neq y$. For $x, y \in X$ with $x \sqsubset y$ the sets $(x, y) = \{z \in X \mid x \sqsubset z \sqsubset y\}$, $(\leftarrow, y) = \{z \in X \mid z \sqsubset y\}$ and $(\rightarrow, x) = \{z \in X \mid x \sqsubset z\}$ are called *intervals*. The set of all intervals on X generates a topology which is called the *interval topology on X induced by $\sqsubseteq$* [Eng89].

To prove that the intervals generate a topology we apply Theorem B.2, i.e. we have to show that the set of all intervals satisfies properties (B1) and (B2) from above. (B1) holds as $x \in (a, b) \cap (c, d)$ for $a, c \in X \cup \{\leftarrow\}$ and $b, c \in X \cup \{\rightarrow\}$ implies $x \in (max\{a, c\}, min\{b, d\})$ where $max\{\leftarrow, y\} = y$ for $y \in X \cup \{\leftarrow\}$ and min is defined analogously. (B2) is valid since for $x \in X$ we can select an arbitrary $y \in X \setminus \{x\}$ (X has at least two elements) such that either $x \in (\leftarrow, y)$ or $x \in (y, \rightarrow)$ holds.

Definition B.9 (Continuous function.) *A function $f \in X \rightarrow Y$ between two topological spaces X and Y is called continuous if the inverse image of every open set of Y is open in X.*

Note that this definition generalizes the one usually used in computer science: There continuity amounts to the preservation of least upper bounds of increasing chains by monotonous functions on Cpo's. If the Scott topologies induced by the order on the domain and range of the function are used, our topological definition and the definition based on Cpo's are equivalent (see [Win93]).

Theorem B.3 (Topology generated by a function.) *Let f be a function from X to topological space Y and let $\mathcal{B}_Y$ be a base for the topology on Y. Then the topology $\mathfrak{O}_X$ generated by the base $\mathcal{B}_X = \{f^{-1}(B) \mid B \in \mathcal{B}_Y\}$ is the coarsest topology on X which makes f continuous.*

Note that $f^{-1}(B)$ is the inverse image of B under f.

Proof. First, we show that $\mathcal{B}_X$ generates a topology. Let $x \in f^{-1}(B_1) \cap f^{-1}(B_2)$, $B_1, B_2 \in \mathcal{B}_Y$. Hence, $f(x) \in B_1 \cap B_2$. As $\mathcal{B}_Y$ is a base, there is a $B \in \mathcal{B}_Y$ with $f(x) \in B \subseteq B_1 \cap B_2$. Thus, $x \in f^{-1}(B)$, $f^{-1}(B) \in \mathcal{B}_X$ and $f^{-1}(B) \subseteq f^{-1}(B_1) \cap f^{-1}(B_2)$ by some set arithmetic. Furthermore, for every $x \in X$ there is a $f^{-1}(B) \in \mathcal{B}_X$ with $x \in f^{-1}(B)$, because of the respective property of $\mathcal{B}_Y$ on Y. Therefore, Theorem B.2 can be applied and yields that $\mathcal{B}_X$ generates a topology on X.

By the definition of base and continuity it is easy to see that f is continuous w.r.t. the topologies generated by $\mathcal{B}_X$ and $\mathcal{B}_Y$.

Third, we show that $\mathcal{B}_X$ is the coarsest topology that makes f continuous. Let $\mathfrak{O}$ be a topology on X such that f is continuous and let α be an open set w.r.t. the topology generated by $\mathcal{B}_X$, i.e. $\alpha = \bigcup U_J$ for some $U_J \subseteq \mathcal{B}_X$. By definition each $U \in U_J$ is given as $U = f^{-1}(B)$ for some $B \in \mathcal{B}_Y$. By the assumption on $\mathfrak{O}$, f is continuous w.r.t. $\mathfrak{O}$ which implies that these $U \in U_J$ are also open w.r.t. $\mathfrak{O}$. Hence, their union α is also open w.r.t. $\mathfrak{O}$ and correspondingly the topology generated by $\mathcal{B}_X$ is coarser than $\mathfrak{O}$. □

Definition B.10 (Tychonoff topology.) *Let $\{X_i\}_{i \in I}$ be a set of topological spaces. The topology on the product space $X = \Pi_{i \in I} X_i$ which is generated by the base $\mathcal{B}_X = \{\bigcap_{k=1}^{\ell} \pi_{i_k}^{-1}(o_{i_k}) \mid o_{i_k} \in \mathfrak{O}_{X_{i_k}} \wedge i_1, \ldots, i_\ell \in I\}$, where $\mathfrak{O}_{X_i}$ is the topology on X_i and π_i is the projection from X to X_i, is called the* Tychonoff topology *on X.*

Due to the definition given in [Eng89] for topologies generated by *sets* of mappings, the Tychonoff topology is generated by the set of projections $\{\pi_i\}_{i \in I}$ (see also Theorem B.3 for topologies generated by *single* functions).

Theorem B.4 (Subspace topology.) *Let $(X, \mathfrak{O})$ be a topological space and M a subset of X. Taking $\{M \cap U \mid U \in \mathfrak{O}\}$ as the open sets in M yields a topological space. M with this topology is called a subspace of X and the topology is called the* subspace topology *on M.*

Proving that this indeed is a topology on M is easy [Eng89].

B.3 Metric Spaces

Definition B.11 (Metric Space.) *A* metric space *is a pair (X, d) consisting of a nonempty set X and a mapping $d : X \times X \to \mathbb{R}_+$, called a* metric *or a* distance, *which has the following properties:*

(1) $\forall x, y \in X : \quad d(x, y) = 0 \quad \Leftrightarrow \quad x = y$
(2) $\forall x, y \in X : \quad d(x, y) = d(y, x)$
(3) $\forall x, y, z \in X : \quad d(x, y) \leq d(x, z) + d(z, y)$ *(triangle inequality)*

A very simple example of a metric space is the discrete metric space. We use it on sets M different from the real numbers.

Definition B.12 (Discrete metric space.) *The* discrete metric space *(M, ϱ) over a set M is defined as follows:*

$$\forall m_1, m_2 \in M : \quad \varrho(m_1, m_2) = \begin{cases} 0 & \text{if } m_1 = m_2 \\ 1 & \text{if } m_1 \neq m_2. \end{cases}$$

Obviously, ϱ is a metric.

Definition B.13 (Natural metric on the real line.) *The natural metric d on the real line is defined by $d(x, y) = |x - y|$ for all $x, y \in \mathbb{R}$.*

Unless otherwise mentioned, we use this metric on the real numbers and on $\mathbb{R}_+$.

Theorem B.5 (Topology induced by a metric.) *For a metric space (X, d), the set $\mathcal{B} = \bigcup_{x \in X, r > 0} \{y \in X \mid d(x, y) < r\}$ of subsets of X is a base for a topology on X.*

The topology on X generated by this base is called the *topology induced by the metric d.*

Proof. A proof is given in [Eng89]. □

Definition B.14 (Natural topology on the real line.) *The topology on $\mathbb{R}$ which is induced by the natural metric on the real line is called the* natural topology on $\mathbb{R}$.

Definition B.15 (Lipschitz continuity.) *Let (X_1, d_1) and (X_2, d_2) be metric spaces and let $f \in X_1 \to X_2$ be a function. We call f* Lipschitz continuous with Lipschitz constant $L \geq 0$ *if the following condition is satisfied: $\forall x, y \in X_1. \quad d_2(f(x), f(y)) \leq L \cdot d_1(x, y)$.*
Function f is called Lipschitz continuous *if there exists an $L \geq 0$ such that f is Lipschitz continuous with Lipschitz constant L.*

Note that Lipschitz continuity for dense streams $\sigma \in M^{\mathbb{R}_{p+}}$ whose range M is a discrete metric space means that they must be constant. The proof proceeds by contradiction. Assume $L \geq 0$ is a Lipschitz constant. If σ is not constant, there must be a point x whose environment $\{x' \mid d(x,x') < \frac{1}{L}\}$ contains a point y with $\sigma(x) \neq \sigma(y)$. Thus, $L \cdot d(x,y) < 1 = \varrho(\sigma(x),\sigma(y))$ holds, where ϱ is the discrete metric. This contradicts Lipschitz continuity.

B.4 Convergence in Metric Spaces

Definition B.16 (Convergence.) *A sequence of elements (a_i) in a metric space (D,d) is* converging *to limit $a \in D$, if for every $\varepsilon > 0$ there exists a number $k \in \mathbb{N}$ such that $d(a_i,a) \leq \varepsilon$ for all $i \geq k$ [Eng89].*

Due to the properties of metrics, any sequence in a metric space has at most one limit.

The following yields a helpful technique for proving convergence of a sequence.

Definition B.17 (Cauchy Sequence.) *For a metric space (D,d) a sequence of elements (a_i) in D is called a* Cauchy sequence*, if for every $\varepsilon > 0$ there exists a number $k \in \mathbb{N}$ such that $d(a_i,a_k) \leq \varepsilon$ for all $i \geq k$ [Eng89].*

Obviously, any convergent sequence in a metric space is a Cauchy sequence.

Definition B.18 (Complete Metric Space.) *A metric space (D,d) is called* complete*, if every Cauchy sequence in D converges to an element in D [Eng89].*

Theorem B.6 (Convergence and closed sets) *A set M is closed iff it contains the limit of every converging sequence (m_i) in M.*

Proof. A proof of a more general theorem is given in [Eng89]. □

B.5 The Metric Space of Streams

For infinite dense and discrete-time streams, i.e. for time model $\mathcal{T} \in \{\mathbb{R}_+, \mathbb{N}\}$, we use the following metric:

Definition B.19 (The Metric Space of Streams.) *The* metric space of streams *$(X^{\mathcal{T}}, d_s)$ is for all $x,y \in X^{\mathcal{T}}$, $X \neq \emptyset$, defined as follows:*

$$d_s(x,y) = inf\{2^{-t} \mid t \in \mathcal{T} \wedge x|_{[0,t)\cap\mathcal{T}} = y|_{[0,t)\cap\mathcal{T}}\}$$

where the infimum of the empty set is defined as $inf\{\} = 1$.

This definition is adapted from [GR95] and [MS97]. Using set $[0,t)\cap\mathcal{T}$ in the restrictions allows us to homogeneously define the metric for time models $\mathbb{R}_+$ and $\mathbb{N}$. The infimum of the empty set has to be defined to ensure well definedness of the metric in cases where the considered streams x and y already differ at time 0. It is easy to prove that d_s is indeed a metric. It closely resembles the Baire metric defined in [Eng89], but is not equivalent to it.

The distance of two streams in the metric d_s encodes the last point in time up to which they coincide. Here, a sequence (a_i) is a Cauchy sequence iff for growing k the length t of the coincident prefixes of streams a_k and a_i, with $i \geq k$, grows without bound.

Theorem B.7 *The metric spaces $(X^{\mathbb{R}_{p+}}, d_s)$ and $(X^{\mathbb{N}}, d_s)$ are complete.*

Proof. Let $\mathcal{T} \in \{\mathbb{R}_+, \mathbb{N}\}$ and let (a_i) be a Cauchy sequence in $X^{\mathbb{R}_{p+}}$ or $X^{\mathbb{N}}$, respectively. We define the fixed point a of (a_i) as follows: For each $t \in \mathcal{T}$ we choose a k such that $\forall i \geq k.\, a_i|_{[0,t)\cap\mathcal{T}} = a_k|_{[0,t)\cap\mathcal{T}}$ and define $a|_{[0,t)\cap\mathcal{T}} = a_k|_{[0,t)\cap\mathcal{T}}$. Such a k exists, because of the definition of the metric of streams and of Cauchy sequences. Clearly, the sequence (a_i) converges to a.

If $\mathcal{T} = \mathbb{N}$ we are done, because a is a function from $\mathcal{T}$ to X. For $\mathcal{T} = \mathbb{R}_+$ it remains to prove that a is piecewise smooth and piecewise Lipschitz continuous to obtain $a \in X^{\mathbb{R}_{p+}}$, i.e. that a is a dense stream. Piecewise smoothness as well as piecewise Lipschitz continuity are defined over finite intervals. Therefore, if a is not piecewise smooth and piecewise Lipschitz there must be a time t such that in $[0,t)$ there is a subinterval in which a can not be partitioned into finitely many smooth and Lipschitz continuous segments. Due to construction this implies that some of the (a_i) must already have failed being in $X^{\mathbb{R}_{p+}}$. Thus, $X^{\mathbb{R}_{p+}}$ also is complete. □

We can extend the metric on streams d_s from infinite to finite *and* infinite streams. For infinite time model $\mathcal{T} \in \{\mathbb{R}_+, \mathbb{N}\}$ and finite or infinite stream $x \in \bigcup_{t>0} X^{[0,t)\cap\mathcal{T}} \cup X^{\mathcal{T}}$, we write $\overline{x}$ to denote its extension to infinite time. We define $\overline{x}$ such that $\overline{x} \in (X \cup \{\bot\})^{\mathcal{T}}$, where $\bot$ is a designated element not in X. For finite time streams $x \in X^{[0,t')}$ the extension is defined by $\overline{x}(t) = x(t)$ if $t < t'$ and $x(t) = \bot$ otherwise. For infinite time streams $x \in X^{\mathcal{T}}$ we define $\overline{x} = x$. Note that $\overline{x} = \overline{y}$ iff $x = y$.

Definition B.20 *The* metric space of finite and infinite streams $(\bigcup_{t>0} X^{[0,t)\cap\mathcal{T}} \cup X^{\mathcal{T}}, d_{\overline{s}})$ *for $X \neq \emptyset$ and $\mathcal{T} \in \{\mathbb{R}_+, \mathbb{N}\}$ results from defining $d_{\overline{s}}(x,y) = d_s(\overline{x},\overline{y})$ for all $x, y \in \bigcup_{t>0} X^{[0,t)\cap\mathcal{T}} \cup X^{\mathcal{T}}$.*

The definition of $\overline{x}$ allows to deduce that $d_{\overline{s}}$ is a metric from the respective properties of d_s.

All infinite, converging sequences in the metric space of finite and infinite streams are either constant eventually or their limit is an infinite stream. This holds, because our metric guarantees that two distinct, finite streams of length t_1 and t_2 have as

least distance $2^{-min\{t_1,t_2\}}$. As an example we regard the sequence (a_i), where each a_i is constantly 0 and $a_i \in \{0\}^{[0,t_i)}$, with $t_i = \Sigma_{k=0}^{i} \frac{1}{2}^k$. For $i \to \infty$, t_i converges to 2, i.e. no infinite stream results from this sequence. As (a_i) furthermore is not constant eventually, it does not converge w.r.t. the metric space of dense streams. In particular, $d_{\overline{s}}(a, a_i) > \frac{1}{4}$ for the constant function $a \in \{0\}^{[0,2]}$ and $i \in \mathbb{N}$.

Theorem B.8 *The metric spaces of finite and infinite dense streams and the metric space of finite and infinite discrete time streams are complete.*

Proof. Let (a_i) be a Cauchy sequence in the metric spaces of finite and infinite dense streams or the metric space of finite and infinite discrete time streams. If the sequence is constant eventually, the claim is trivial. Otherwise we proceed similar to the proof of Theorem B.7: Let $\mathcal{T} = \mathbb{R}_+$ in case of dense streams and $\mathcal{T} = \mathbb{N}$ in case of discrete-time streams. For each $t \in \mathcal{T}$ we choose a k such that $\forall i \geq k. a_i \in X^{[0,t)\cap\mathcal{T}} \wedge a_i|_{[0,t)\cap\mathcal{T}} = a_k|_{[0,t)\cap\mathcal{T}}$ and define $a|_{[0,t)\cap\mathcal{T}} = a_k|_{[0,t)\cap\mathcal{T}}$. Such a k exists, because of the definition of the metric of streams, which also guarantees that two distinct, finite streams of length t and t' have as least distance $2^{-min\{t,t'\}}$, and because of the definition of Cauchy sequences. Clearly, the sequence (a_i) converges to a.

For dense streams we furthermore have to show that a is piecewise smooth and piecewise Lipschitz continuous. Here, the argument is the same as in the proof of Theorem B.7. □

The following theorem relates time-divergent prefix monotonous sequences (Definition A.1) to Cauchy sequences.

Theorem B.9 *Every time-divergent prefix monotonous sequence also is a Cauchy sequence.*

Proof. The proof is straightforward by applying the definitions. The idea is that the coincident prefix of streams a_k and a_i, with $i \geq k$, in a time-divergent prefix monotonous sequence (a_i) grows without bound for growing k. □

If follows that time-divergent prefix monotonous sequences converge in the metric space of finite and infinite dense streams.

B.5.1 Divergence Closure and Topological Closure

In Section A.1.1 divergence closure is defined based on *time-divergent prefix monotonous sequences* (Definition A.1), which are a special case of converging sequences in the metric space of finite and infinite streams.

Theorem B.10 (Topological closure implies divergence closure.) *Let $M_t \subseteq X^{[0,t)}$ for all $t > 0$ and $M \subseteq X^{\mathbb{R}_{p+}}$ be sets of dense streams. If $\bigcup_{t>0} M_t \cup M$ is*

closed w.r.t. the metric space of finite and infinite dense streams then $\bigcup_{t>0} M_t \cup M$ *is divergence closed.*

Proof. Let (m_i) be a time-divergent prefix monotonous sequence with $m_i \in M_{t_i}$ for all i and diverging time sequence (t_i). Let m be the limit of (m_i). Closure of $\bigcup_{t>0} M_t \cup M$ guarantees that m is in $\bigcup_{t>0} M_t \cup M$ (Theorem B.6). As the limit of t_i is ∞, limit m is an infinite stream and hence $m \in M$. □

In general, divergence closure does not imply topological closure, because divergence closure does not make any statements about the limits of sequences of streams in M (and not in M_t), whereas topological closure demands that the limit of such a sequence is in M. Equivalence would hold, if M_t is defined as $M_t = M|_{[0,t)}$, because in this case for a sequence in M which converges, we can construct a time-divergent prefix monotonous sequence in M_t.

Bibliography

[ABHL97] T. Amon, G. Borriello, T. Hu, and J. Liu. Symbolic timing verification of timing diagrams using Presburger formulas. In *Proceedings of the 34th Design Automation Conference*, pages 226–231. ACM, 1997.

[Abr96] J.-R. Abrial. Steam-boiler control specification problem. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS 1165. Springer-Verlag, 1996.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[AGH+00] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in Charon. In *Proc. of 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, LNCS 1790. Springer-Verlag, 2000.

[AGLS01] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional refinement of hierarchical hybrid systems. In *Proc. of 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, LNCS 2034. Springer-Verlag, 2001.

[AH97] R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR 97: Concurrency Theory*, Lecture Notes in Computer Science 1243, pages 74–88. Springer-Verlag, 1997.

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.

[AHS96] R. Alur, T. A. Henzinger, and E. D. Sontag, editors. *Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[Aki93] E. Akin. *The General Topology of Dynamical Systems*. American Mathematical Society, 1993.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[Ali99] L. O. Alima. *Self-Stabilization by Self-Stabilizing Waves*. PhD thesis, Université catholique de Louvain, Faculté des Sciences Appliquées, Département d'Ingéniere Informatique, Belgium, 1999.

[Ant95] P. J. Antsaklis, editor. *Hybrid systems II*, LNCS 999. Springer-Verlag, 1995.

[Ant96] F. Antosch. *Systematische Maschinenmodellierung: Modellierung eines Glideliners (Fa. Krones) nach der ROOM Methode*. Diplomarbeit, Lehrstuhl für Informationstechnik im Maschinenwesen, Technische Universität München, 1996.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[AT98] J. Albert and J. Tomaszunas. Komponentenbasierte Modellbildung und Echtzeitsimulation kontinuierlich-diskreter Prozesse. In *Proc. of VDI/VDE GMA Kongreß Meß- und Automatisierungstechnik*, Ludwigsburg, 1998.

[Bal98] H. Balzert. *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, Heidelberg, 1998.

[Ber96] G. Berry. The constructive semantics of Esterel. Book in preparation, 1996.

[Bey01] D. Beyer. Reachability analysis and refinement checking for BDD-based model checking of timed automata. Technical report, Technical University Cottbus, 2001.

[BG90] A. Benveniste and P. Le Guernic. Hybrid dynamical systems theory and the Signal language. *IEEE Transactions on Automatic Control*, 35(5):535–546, May 1990.

[BGM93] A. Back, J. Guckenheimer, and M. Myers. A dynamical simulation facility for hybrid systems. In *Proc. of Hybrid Systems I*, LNCS 736. Springer-Verlag, 1993.

[BHKT98] H. Brettschneider, H.-M. Hanisch, S. Kowalewski, and J. Thieme. Diskontinuierlicher Verdampfer: Ein erweitertes Benchmark-Beispiel für ereignisdiskrete und hybride Systeme. Presentation at the 1998 Workshop of GMA Fachausschuß 1.8 "Methoden der Steuerungstechnik", March 1998.

[BHS99] M. Broy, F. Huber, and B. Schätz. AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. *Informatik Forschung und Entwicklung*, 14(3):121–134, 1999.

[BLL+96] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal in 1995. In *Proc. of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, LNCS 1055, pages 431–434. Springer-Verlag, 1996.

[BR95] K. Buchenrieder and J. Rozenblit. Codesign: An overview. In J. Rozenblit and K. Buchenrieder, editors, *Codesign – Computer-aided Hardware/Software Engineering*, pages 1–15. IEEE Press, 1995.

[Bra94] M. S. Branicky. Stability of switched and hybrid systems. In *Proc. 33rd IEEE Conf. Decision and Control*, pages 3498–3503, Lake Buena Vista, FL, 1994.

[Bro93] M. Broy. (Inter-)action refinement: The easy way. In *Program Design Calculi, NATO ASI*, volume 118 of *Series F: Computer and Systems Sciences*. IOS Press, 1993.

[Bro97a] J. F. Broenink. Modelling, simulation and analysis with 20-Sim. *Journal A, the special issue on CACSD*, 97(3):22–25, 1997.

[Bro97b] M. Broy. Refinement of time. In *Proc. of ARTS'97*, LNCS 1231. Springer-Verlag, 1997.

[Bro97c] M. Broy. Requirements engineering for embedded systems. In *Proc. of FemSys'97*, München, April 1997.

[Bro99a] J. F. Broenink. Introduction to physical systems modelling with bond graphs. To be published in the SiE Whitebook on Simulation methodologies, http://www.rt.el.utwente.nl/bnk/papers/BondGraphsV2.pdf, 1999.

[Bro99b] M. Broy. A logical basis for modular systems engineering. In *Proc. of the 1998 NATO ASI International Summer School on Calculational System Design*, volume 173 of *Series F: Computer and Systems Sciences*. IOS Press, 1999.

[Bro01] M. Broy. Refinement of time. *Theoretical Computer Science*, 253(1):3–26, 2001.

[BS99] M. Broy and T. Stauner. Requirements Engineering für eingebettete Systeme. *Informationstechnik und technische Informatik (it+ti)*, 41(2):7–11, 1999.

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems:* FOCUS *on Streams, Interfaces and Refinement*. Springer-Verlag, 2001.

[BSMM97] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, third edition, 1997.

[Cas97] P. Caspi. What can we learn from synchronous data-flow languages? In *Proc. Int. Workshop on Hybrid and Real-Time Systems (HART'97)*, LNCS 1201. Springer-Verlag, 1997.

[CB99] E. Christen and K. Bakalar. VHDL-AMS – a hardware description language for analog and mixed-signal applications. *IEEE Transactions on CAS-II*, 46(10):1263–1272, 1999.

[CC92] P. Cousot and R. Cousot. Abstract Interpretation and applications to logic programs. *Journal of Logic Programming*, 13(2-3):103–180, 1992.

[CG92] M. A. Celia and W. G. Gray. *Numerical Methods for Differential Equations*. Prentice Hall, 1992.

[CH99] H. Chen and H.-M. Hanisch. Control synthesis of hybrid systems based on predicate invariance. In *Proc. of Hybrid Systems V*, LNCS 1567. Springer-Verlag, 1999.

[CHR92] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 30:269–276, 1992.

[CMP91] E. Chang, Z. Manna, and A. Pnueli. The safety-progress classification. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specifications*, NATO ASI. Springer-Verlag, 1991.

[Cou90] P. Cousot. Methods and logics for proving programs. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier, 1990.

[CRH93] Z. Chaochen, A. P. Ravn, and M. R. Hansen. An extended duration calculus for hybrid real-time systems. In *Hybrid Systems*, LNCS 736. Springer-Verlag, 1993.

[CWM98] M. Conrad, M. Weber, and O. Müller. Towards a methodology for the design of hybrid systems in automotive electronics. In *Proc. of the International Symposium on Automotive Technology and Automation (ISATA'98)*, 1998.

[Dai01] DaimlerChrysler. DaimlerChrysler Vertriebsorganisation Deutschland CL-Klasse Coupé – Distronic. http://www.mercedes-benz.de/mbd/t03/0,1316,C24_2RD,00.html, 2001.

[dAM95] L. de Alfaro and Z. Manna. Verification in continuous time by discrete reasoning. In *Proc. of 4th International Conference on Algebraic Methodology and Software Technology (AMAST)*, LNCS 936. Springer-Verlag, 1995.

[DBSV01] M.D. Di Benedetto and A.L. Sangiovanni-Vincentelli, editors. *Proceedings of the Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01),* Rome, Italy, LNCS 2034. Springer-Verlag, 2001.

[DFG01] DFG. Schwerpunktprogramm KONDISK (Analyse und Synthese kontinuierlich-diskreter Systeme). http://www.ifra.ing.tu-bs.de/kondisk/, 2001.

[Die96] C. Dietz. Graphical formalization of real-time requirements. In B. Jonsson and J. Parrow, editors, *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96)*, LNCS 1135, pages 366–385. Springer-Verlag, 1996.

[Dij82] E. W. Dijkstra. Self-stabilization in spite of distributed control (EWD 391). In E. W. Dijkstra, editor, *Selected Writings on Computing: A Personal Perspective*, Texts and Monographs in Computer Science. Springer-Verlag, 1982.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Proc. of Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[DSG+94] S. Donnay, K. Swings, G. Gielen, W. Sansen, W. Kruiskamp, and D. Leenaerts. A Methodology for Analog Design Automation in Mixed-Signal ASICs. In *Proc. of The European Design Automation Conference (EURO-DAC'94)*, pages 530–534. IEEE Computer Society Press, 1994.

[DW98] D. F. D'Souza and A. C. Wills. *Objects, components and frameworks with UML - the Catalysis approach.* Addison-Wesley Object Technology Series, 1998.

[EH96] S. Engell and I. Hoffmann. Modular hierarchical models of hybrid systems. In *Proc. of the 35th IEEE Conference on Decision and Control (CDC)*, pages 142–143, Kobe, 1996.

[EKM+93] M. Engel, M. Kubica, J. Madey, D. L. Parnas, A. P. Ravn, and A. J. van Schouwen. A formal approach to computer systems requirements documentation. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736. Springer-Verlag, 1993.

[Eme90] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier, 1990.

[Eng89] R. Engelking. *General Topology*, volume 6 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1989.

[Eng97] S. Engell. Modellierung und Analyse hybrider dynamischer Systeme. *at – Automatisierungstechnik*, 45(4), 1997.

[FEM+98] M. Fuchs, M. Eckrich, O. Müller, J. Philipps, and P. Scholz. Advanced design and validation techniques for electronic control units. In *Proc. of the International Congress of the Society of Automotive Engineers*. SAE International, 1998.

[FKMF97] F. Fischer, T. Kolloch, A. Muth, and G. Färber. A configurable target architecture for rapid prototyping high performance control systems. In *Proc. of the Int. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, volume 3, pages 1382–1390, Las Vegas, Nevada, USA, 1997.

[Flo67] R. W. Floyd. Assigning meaning to programs. In *Mathematical aspects of computer science: Proc. American Mathematics Society Symposia*, volume 19. American Mathematical Society, 1967.

[FNW98] V. Friesen, A. Nordwig, and M. Weber. Object-oriented specification of hybrid systems using UML^h and ZimOO. In *Proc. 11th Int. Conf. on the Z Formal Method (ZUM)*, LNCS 1493. Springer-Verlag, 1998.

[Föl87] O. Föllinger. *Nichtlineare Regelungen I*. R. Oldenbourg Verlag, München, fourth edition, 1987.

[Föl90] O. Föllinger. *Regelungstechnik*. Hüthig Buch Verlag, Heidelberg, sixth edition, 1990.

[Frä99] M. Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In *Proc. of Computer Science Logic (CSL'99)*, LNCS 1683. Springer-Verlag, 1999.

[Fri86] B. Friedland. *Control System Design – an introduction to state-space methods*. McGraw-Hill Inc., 1986.

[Fri98a] V. Friesen. A logic for the specification of continuous systems. In *Proc. of Hybrid Systems: Computation and Control (HSCC'98)*, LNCS 1386. Springer-Verlag, 1998.

[Fri98b] V. Friesen. *Objektorientierte Spezifikation hybrider Systeme.* GMD-Bericht Nr. 289, Oldenbourg Verlag, 1998.

[FvBR98] G. Fábián, D. A. van Beek, and J. E. Rooda. Integration of the discrete and the continuous behaviour in the hybrid chi simulator. In *1998 European Simulation Multiconference, Manchester*, pages 252–257, 1998.

[GBSS98] R. Grosu, M. Broy, B. Selic, and Gh. Ştefănescu. Towards a calculus for UML-RT specifications. In *7th OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, Technical Report TUM-I9820. Technische Universität München, 1998.

[GBSS99] R. Grosu, M. Broy, B. Selic, and Gh. Ştefănescu. What is behind UML-RT? In *Behavioral specifications of businesses and systems.* Kluwer Academic Publishers, 1999.

[Ger97] S. Germann. *Modellbildung und modellgestützte Regelung der Fahrzeuglängsdynamik.* Fortschrittsberichte VDI, Reihe 12: Verkehrstechnik/Fahrzeugtechnik, Bericht 307. VDI Verlag, 1997.

[Geu96] F. Geurts. *Compositional Analysis of Iterated Relations: Dynamics and Computations.* PhD thesis, Université catholique de Louvain, Faculté des Sciences Appliquées, Département d'Ingéniere Informatique, Belgium, 1996.

[GHJ97] V. Gupta, T.A. Henzinger, and R. Jagadeesan. Robust timed automata. In O. Maler, editor, *HART 97: Hybrid and Real-time Systems*, Lecture Notes in Computer Science 1201, pages 331–345. Springer-Verlag, 1997.

[GKS00] R. Grosu, I. Krüger, and T. Stauner. Hybrid Sequence Charts. In *Proc. of the 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2000).* IEEE, 2000.

[GNRR93] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems I*, LNCS 736. Springer-Verlag, 1993.

[GPV94] G. Göllü, A. Puri, and P. Varaiya. Discretization of timed automata. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 957–958, 1994.

[GR95] R. Grosu and B. Rumpe. Concurrent timed port automata. Technical Report TUM-I9533, Technische Univerität München, 1995.

[Gri99] C. Grimm. *Hybride Datenflussgraphen und ihre Anwendung beim Entwurf analog/digitaler Systeme.* PhD thesis, J. W. Goethe-Universität, Frankfurt am Main, 1999.

[Gro00] Object Management Group. Unified Modeling Language (UML), version 1.3. http://www.omg.org/technology/documents/new_formal/unified_modeling_language.htm, 2000.

[GS00a] C. Grimm and T. Stauner. Übersetzung von HyCharts in HDFG. In *Proc. of 3. ITG/GI/GMM-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Forschungs-Report. VDE Verlag, 2000.

[GS00b] R. Grosu and T. Stauner. Modular and visual specification of hybrid systems – an introduction to HyCharts. Accepted for *Formal Methods in System Design*, 2000.

[GSB98a] R. Grosu, T. Stauner, and M. Broy. A modular visual model for hybrid systems. In *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, LNCS 1486. Springer-Verlag, 1998.

[GSB98b] R. Grosu, Gh. Ştefănescu, and M. Broy. Visual formalisms revisited. In *Proc. International Conference on Application of Concurrency to System Design (CSD'98)*, 1998.

[GW98] C. Grimm and K. Waldschmidt. Repartitioning and technology mapping of electronic hybrid systems. In *Proceedings of DATE '98*, Paris, February 1998. IEEE.

[Han00] H.-M. Hanisch. Störungsüberwachung und Synthese von Abfahrsteuerungen in hybriden Systemen. http://www.ifra.ing.tu-bs.de/kondisk/hanisch.html, 2000.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, March 1987.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[Hei96] C. Heitmeyer. Requirements specification for hybrid systems. In *Proc. of Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[Hen91] T.A. Henzinger. *The Temporal Specification and Verification of Real-time Systems*. PhD thesis, Stanford University, 1991.

[Hen00] T.A. Henzinger. Masaccio: a formal model for embedded components. In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. Ito, editors, *TCS 00: Theoretical Computer Science*, Lecture Notes in Computer Science 1872, pages 549–563. Springer-Verlag, 2000.

[HG96] D. V. Hung and P. H. Giang. Sampling semantics of duration calculus. In *Proceedings of FTRTFT '96*, LNCS 1135. Springer-Verlag, 1996.

[HH91] G. Hämmerlin and K.-H. Hoffmann. *Numerische Mathematik*. Springer-Verlag, Berlin, second edition, 1991.

[HHMW00] T.A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HyTech: hybrid systems analysis using interval numerical methods. In N.A. Lynch and B. Krogh, editors, *HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 130–144. Springer-Verlag, 2000.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1019, pages 41–71. Springer-Verlag, 1995.

[HMP93] T.A. Henzinger, Z. Manna, and A. Pnueli. Towards refining temporal specifications into hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, Lecture Notes in Computer Science 736, pages 60–76. Springer-Verlag, 1993.

[HMP01] T.A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In M. di Benedetto and A. Sangiovanni-Vincentelli, editors, *HSCC 01: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2034, pages 275–290. Springer-Verlag, 2001.

[Hoa86] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1986.

[HR00] T.A. Henzinger and J.-F. Raskin. Robust undecidability of timed and hybrid systems. In N.A. Lynch and B. Krogh, editors, *HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 145–159. Springer-Verlag, 2000.

[HS97] A. Hussein and D. Sinclair. Design methodology for hybrid systems. Technical Report CA-1897, Dublin City University, 1997.

[HS98] T.A. Henzinger and S. Sastry, editors. *HSCC 98: Hybrid Systems—Computation and Control*. Lecture Notes in Computer Science 1386. Springer-Verlag, 1998. Proceedings of the First International Workshop.

[HSTV00] J. Haase, P. Schwarz, P. Trappe, and W. Vermeiren. Erfahrungen mit VHDL-AMS bei der Simulation heterogener Systeme. In *Proc. of 3. ITG/GI/GMM-Workshop Methoden und Beschreibungssprachen zur*

Modellierung und Verifikation von Schaltungen und Systemen. VDE Verlag, 2000.

[Int98] Integrated Systems Inc. Integrated Systems: Products - BetterState. http://www.isi.com/Products/BetterState/, 1998.

[Int00] Integrated Systems Inc. Integrated Systems: Product families - MATRIXx. http://www.isi.com/Products/MATRIXx/, 2000.

[IT96] ITU-TS. Recommendation Z.120 : Message Sequence Chart (MSC). Geneva, 1996.

[ITU94] ITU. ITU-T Recommendation Z.100: CCITT Specification and Description Language (SDL), June 1994.

[Jai79] M. K. Jain. *Numerical Solution of Differential Equations*. Wiley Eastern Ltd., New Delhi, 1979.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. North-Holland Publishing Company, 1974.

[KFS95] M. Kloas, V. Friesen, and M. Simons. Smile — a simulation environment for energy systems. In *Proc. of the 5th International IMACS-Symposium on Systems Analysis and Simulation (SAS'95)*, Systems Analysis Modelling Simulation, volume 18-19, pages 503–506. Gordon and Breach Publishers, 1995.

[KL94] W. Kortüm and P. Lugner. *Systemdynamik und Regelung von Fahrzeugen*. Springer-Verlag, 1994.

[Kön90] K. Königsberger. *Analysis I*. Springer-Verlag, 1990.

[KP92] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems 2nd International Symposium*, LNCS 571, pages 591–619. Springer-Verlag, 1992.

[Krü00] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[Lam93] L. Lamport. Hybrid systems in TLA+. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Proc. of Hybrid Systems*, LNCS 736. Springer-Verlag, 1993.

[Lee01] E. A. Lee. Overview of the Ptolemy project. Technical Report Technical Memorandum UCB/ERL M01/11, University of California, Berkeley, 2001.

[Lei87] H. Leipholz. *Stability Theory*. John Wiley & Sons Ltd., second edition, 1987.

[LK00] N. A. Lynch and B. Krogh, editors. *Proc. of 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, LNCS 1790. Springer-Verlag, 2000.

[LNR97] J. Lunze, B. Nixdorf, and H. Richter. Hybrid modelling of continuous-variable systems with application to supervisory control. In *Proc. of European Control Conference 1997*, 1997.

[LSVW96] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[Lue79] D. G. Luenberger. *Introduction to Dynamic Systems – Theory, Models, and Applications*. John Wiley & Sons Ltd., 1979.

[Mai87] M. Main. Trace, failure and testing equivalences for communicating processes. *Int. Journal of Parallel Programming*, 16(5):383–400, 1987.

[Mar91] F. Maraninchi. The Argos language: Graphical representation of automata and description of reactive systems. In *IEEE Workshop on Visual Languages*, oct 1991.

[Mer98] S. Merz. On the verification of a self-stabilizing algorithm. Available at http://www.pst.informatik.uni-muenchen.de/~merz/papers/dijkstra.ps.gz, 1998.

[MF00] A. Muth and G. Färber. SDL as a system level specification language for application-specific hardware in a rapid prototyping environment. In *Proc. of the 13th Int. Symposium on System Synthesis (ISSS'2000)*, Madrid, Spain, 2000. IEEE Computer Society Press.

[Mis96] M. W. Mislove. Topology, domain theory and theoretical computer science. Preprint, available at http://www.math.tulane.edu/ftp/pub/mwm/topandcs.ps.gz, 1996.

[MOE99] S. Mattsson, M. Otter, and H. Elmqvist. Modelica hybrid modeling and efficient simulation. In *Proc. of 38th IEEE Conference on Decision and Control (CDC99)*. IEEE, 1999.

[Mos97] P. J. Mosterman. *Hybrid Dynamic Systems: A Hybrid Bond Graph Modeling Paradigm and its Application in Diagnosis.* PhD thesis, Department of Electrical Engineering, Vanderbilt University, 1997.

[Mos99] P. J. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In *Hybrid Systems Computation and Control (HSCC'99)*, LNCS 1569. Springer-Verlag, 1999.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[MS97] O. Müller and P. Scholz. Functional specification of real-time and hybrid systems. In *Proc. Int. Workshop on Hybrid and Real-Time Systems (HART'97)*, LNCS 1201. Springer-Verlag, 1997.

[MS00] O. Müller and T. Stauner. Modelling and verification using linear hybrid automata - a case study. *Mathematical and Computer Modelling of Dynamical Systems (MCMDS)*, 6(1):71–89, 2000.

[MT75] M. D. Mesarovic and Y. Takahara. *General Systems Theory: Mathematical Foundations*, volume 113 of *Mathematics in Science and Engineering.* Academic Press, 1975.

[MW95] A. N. Michel and K. Wang. *Qualitative theory of dynamical systems – The role of stability preserving mappings.* Marcel Dekker Inc., 1995.

[NNH98] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer-Verlag, 1998.

[Oga87] K. Ogata. *Discrete-Time Control Systems.* Prentice-Hall, 1987.

[Pau87] Lawrence C. Paulson. *Logic and Computation.* Cambridge University Press, 1987.

[Pay61] H. M. Paynter. *Analysis and Design of Engineering Systems.* MIT press, 1961.

[PH88] C. L. Phillips and R. D. Harbor. *Feedback Control Systems.* Prentice-Hall, 1988.

[PM95] D. L. Parnas and J. Madey. Functional documents for computer systems. *Science of Computer Programming*, 25(1):41–61, October 1995.

[PMK+99] S. Petters, A. Muth, T. Kolloch, T. Hopfner, F. Fischer, and G. Färber. The REAR framework for emulation and analysis of embedded hard real-time systems. In *Proc. of the 10th IEEE Int. Workshop on Rapid Systems Prototyping (RSP'99)*, pages 100–107. IEEE Computer Society Press, 1999.

[Pnu94] A. Pnueli. Development of hybrid systems. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, LNCS 863. Springer-Verlag, 1994.

[PPS00] I. Péter, A. Pretschner, and T. Stauner. Heterogeneous development of hybrid systems. In *Proc. of GI workshop Rigorose Entwicklung softwareintensiver Systeme*. Ludwig-Maximilians-Universität München, Institut für Informatik, Bericht 0005, September 2000.

[PR97] J. Philipps and B. Rumpe. Refinement of information flow architectures. In *Proc. of ICFEM'97, Hiroshima. Japan.* IEEE CS Press, 1997.

[PR99] J. Philipps and B. Rumpe. Refinement of pipe-and-filter architectures. In J. M. Wing, J. Woodcock, and J. Davies, editors, *Proc. of the World Congress on Formal Methods in the Development of Computing System (FM'99)*, LNCS 1708. Springer-Verlag, 1999.

[PS97] J. Philipps and P. Scholz. Compositional specification of embedded systems with Statecharts. In *Proc. of Theory and Practice of Software Development (TAPSOFT'97)*, LNCS 1214. Springer-Verlag, 1997.

[PSS00] A. Pretschner, O. Slotosch, and T. Stauner. Developing correct safety critical, hybrid, embedded systems. In *Proc. of New Information Processing Techniques for Military Systems*, RTO Meeting Proceedings MP-049. NATO Research and Technology Organization, October 2000.

[Rai98] J. Raisch. A hierarchy of discrete abstractions for a hybrid plant. *JESA - European Journal of Automation, special issue on Hybrid Dynamical Systems*, 32:1073–1095, 1998.

[Rap98] M. Rappl. Strukturierte Analyse und Design hybrider Systeme mit MatrixX und BetterState. Diplomarbeit, Fakultät für Informatik, Technische Universität München, 1998.

[RO98] J. Raisch and S. D. O'Young. Discrete approximation and supervisory control of continuous systems. *IEEE Transactions on Automatic Control, Special Issue on Hybrid Systems*, 43(4):569–573, 1998.

[RS98] M. Rönkkö and K. Sere. Refinement and continuous behaviour. Technical Report TUCS No 198, Turku Centre for Computer Science, 1998.

[RTC92] RTCA Inc., EUROCAE. Software considerations in airborne systems and equipment certification. DO-178B / ED-12B, 1992.

[Rum96] B. Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme*. Ph.D. thesis (in German), Technische Universität München, 1996.

[SB98] T. Schlegl and M. Buss. Hybrid closed-loop control of robotic hand regrasping. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 3026–3032, Leuven, Belgium, 1998.

[SBS97] T. Schlegl, M. Buss, and G. Schmidt. Development of numerical integration methods for hybrid (discrete-continuous) dynamical systems. In *Proc. of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM'97)*, Tokyo, Japan, 1997.

[Sch88] H. R. Schwarz. *Numerische Mathematik*. B. G. Teubner, Stuttgart, second edition, 1988.

[Sch98] P. Scholz. *Design of Reactive Systems and their Distributed Implementation with Statecharts*. PhD thesis, Technische Universität München, 1998.

[Sch01] P. Scholz. Incremental design of statechart specifications. *Science of Computer Programming*, 40(1):119–145, 2001.

[SDW95] K. Stølen, F. Dederichs, and R. Weber. Specification and refinement of networks of asynchronously communicating agents using the assumption/commitment paradigm. *Formal Aspects of Computing*, 1995.

[SG90] D. Scott and C. Gunter. Semantic domains and denotational semantics. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 12, pages 633 – 674. Elsevier Science Publishers, 1990.

[SG95] M. Sintzoff and F. Geurts. Analysis of dynamical systems using predicate transformers: Attraction and composition. In *Analysis of Dynamical and Cognitive Systems*, LNCS 888. Springer-Verlag, 1995.

[SG99] T. Stauner and C. Grimm. Prototyping of hybrid systems - from HyCharts to Hybrid Data-Flow Graphs. In *Proc. of WDS'99 (satellite workshop to the 12th International Symposium on Fundamentals of Computation Theory, FCT'99)*, Electronic Notes in Theoretical Computer Science 28. Elsevier Science, 1999.

[SGW94] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons Ltd, Chichester, 1994.

[SHB+99] T. Schlegl, S. Haidacher, M. Buss, F. Freyberger, F. Pfeiffer, and G. Schmidt. Compensation of discrete contact state errors in regrasping experiments with the TUM-hand. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems IROS*, pages 118–123, Kyongju, Korea, 1999.

[Sin92] M. Sintzoff. Invariance and contraction by infinite iterations of relations. In *Research Directions in High-Level Parallel Programming Languages*, LNCS 574. Springer-Verlag, 1992.

[Sin96] M. Sintzoff. Abstract verification of structured dynamical systems. In *Proc. of Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[Sin97] D. Sinclair. Using an object-oriented methodology to bring a hybrid system from initial concept to formal definition. In *Proc. Int. Workshop on Hybrid and Real-Time Systems (HART'97)*, LNCS 1201. Springer-Verlag, 1997.

[SK97] K. Schneider and T. Kropf. The C@S system: Combining proof strategies for system verification. In *Formal Hardware Verification – Methods and Systems in Comparison*, LNCS 1287. Springer-Verlag, 1997.

[Slo98] O. Slotosch. Overview over the project Quest. In *Proc. of Applied Formal Methods – FM Trends 98*, LNCS 1641. Springer-Verlag, 1998.

[SMF97] T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In *Proc. Int. Workshop on Hybrid and Real-Time Systems (HART'97)*, LNCS 1201, pages 139–153. Springer-Verlag, 1997.

[Smi97] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[Son90] E. D. Sontag. *Mathematical Control Theory*. Springer-Verlag, 1990.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2nd edition, 1992.

[SPP01] T. Stauner, A. Pretschner, and I. Péter. Approaching a hybrid UML-RT – tool support and formalization. Paper draft, to appear as Technical Report, Technische Universität München, 2001.

[SRS99] T. Stauner, B. Rumpe, and P. Scholz. Hybrid system model. Technical Report TUM-I9903, Technische Universität München, 1999.

[SSH99] T. Stauner, K. Schneider, and M. Huhn. Translating a visual description technique to a synchronous language: From DiCharts to PURR. In *FBT'99, 9. GI/ITG-Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme"*. Herbert Utz Verlag Wissenschaft, 1999.

[Şte94] G. Ştefănescu. Algebra of flownomials. Technical Report TUM-I9437, Technische Universität München, 1994.

[Şte00] G. Ştefănescu. *Network Algebra*. Springer-Verlag, 2000.

[Sta97] T. Stauner. Specification and verification of an electronic height control system using hybrid automata. Diploma thesis, Technische Universität München, 1997.

[Sta99] T. Stauner. Specification of (parts of) a lip-sync protocol using HyCharts. In *FBT'99, 9. GI/ITG-Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme"*. Herbert Utz Verlag Wissenschaft, 1999.

[Sta00a] T. Stauner. Extending HyCharts with state-invariants. In *Proc. of GI workshop Rigorose Entwicklung software-intensiver Systeme*. Ludwig-Maximilians-Universität München, Institut für Informatik, Bericht 0005, September 2000.

[Sta00b] T. Stauner. Properties of hybrid systems - a computer science perspective. Technical Report TUM-I0017, Technische Universität München, 2000.

[Sta01] T. Stauner. Hybrid systems' properties - classification and relation to computer science. In *Proc. of EuroCAST'2001 (Eight International Conference on Computer Aided Systems Theory)*, LNCS (to appear). Springer-Verlag, 2001.

[Swe95] W. Sweet. The glass cockpit. *IEEE Spectrum*, pages 30–38, September 1995.

[TG00] F. Terrier and S. Gérard. Real time system modeling with UML: current status and some prospects. In *Proc. of the 2nd Workshop on SDL and MSC (SAM2000)*, Grenoble, France, June 2000.

[Tho90] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier, 1990.

[TMI99] The MathWorks Inc. Stateflow. http://www.mathworks.com/products/stateflow/, 1999.

[TMI00] The MathWorks Inc. MATLAB. http://www.mathworks.com/products/matlab/, 2000.

[Toe96] S. Toepper. Rechnergestütztes Lern- und Lehrprogramm "Painless Mechatronics". Jahresbericht, Mechatronik Laboratorium Paderborn (MLaP), 1996.

[Uns00] M. Unser. Sampling – 50 years after Shannon. *Proc. of the IEEE*, 88(4):569–587, 2000.

[Vac95] R. J. Vaccaro. *Digital Control*. McGraw-Hill Inc., 1995.

[vdB00] M. v. d. Beeck. Behaviour specifications: Semantics, equivalence and refinement. In *Proc. of GROOM 2000 (Grundlagen Objekt-Orientierter Modellierung)*. Institut für Informatik, WWU Münster, 2000.

[VvS99] F. W. Vaandrager and J. H. van Schuppen, editors. *Hybrid Systems: Computation and Control II*, LNCS 1569. Springer-Verlag, 1999.

[Wec92] W. Wechler. *Universal Algebra for Computer Scientists*. Springer-Verlag, Berlin, 1992.

[WFSE96] K. Wöllhaf, M. Fritz, C. Schulz, and S. Engell. BaSiP - batch process simulation with dynamically reconfigured process dynamics. *Proc. of ESCAPE-6, Supplement to Comp. & Chem. Engineering*, 972(20):1281–1286, 1996.

[Wie96] R. Wieting. Hybrid high-level nets. In *Proceedings of the 1996 Winter Simulation Conference, Coronado, California, USA / Charnes*, pages 848–855, 1996.

[Wig90] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Texts in Applied Mathematics (TAM) 2. Springer-Verlag, 1990.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

[Won67] W. M. Wonham. On pole assignment in multiinput controlable linear systems. *IEEE Transactions on Automatic Control*, 12:660–665, 1967.

[Zei76] B. P. Zeigler. *Theory of Modelling and Simulation*. John Wiley & Sons Ltd., 1976.

List of Figures

1.1 General architecture of hybrid systems. 2
1.2 Chapter dependencies (numbers refer to the chapters). 7

2.1 General structure of a closed-loop control system. 10
2.2 Development of control systems. 10
2.3 Block diagram for the control of a linear time invariant plant in state-space representation. 12
2.4 Example bond graph. 14
2.5 A conventional development process, based on isolated description techniques (indicated in parenthesis). 17
2.6 Coupling of a state machine and block diagrams in tools such as MATLAB/Simulink/StateFlow. 18
2.7 Integrated development process, based on hybrid description techniques (indicated in parenthesis). Note that pure discrete-event parts are subsumed in the hybrid parts. 20

3.1 Principle of the semantics definition for HyCharts. 32
3.2 The EHC: Architecture and a typical evolution. 34
3.3 The EHC's *Control* component. 35
3.4 Component network and hybrid machine computation model. 37
3.5 Classification of the controlled variables. 39
3.6 Composition of activities. 41
3.7 Computing the possible delay. 43
3.8 Graphical representation of a node $n : A \triangleright B$. 45
3.9 The composition operators. 46
3.10 The connectors. 47
3.11 The symmetric feedback. 48

3.12 The multiplicative interpretation. 51
3.13 Visual attachment of arcs in the additive interpretation. 54
3.14 Visual attachment of nodes in the additive interpretation. 56
3.15 Syntax (left) and semantics (right) of a node in a HySChart. 67
3.16 The semantics of preemption. 71
3.17 The EHC's *Control* component (repeated). 74
3.18 Additive hierarchic graph for *Control*. 76
3.19 The *Control* component's analog part. 80
3.20 HySChart for the *Filter* component. 81

4.1 Adding of (ignored) input channels. 90

5.1 Delayed or failing detection of boundary crossing events caused by sampling. 97
5.2 Discretization error and intersample error. 98
5.3 The relaxed invariant remains true for values up to the permitted deviation ε_a. 104
5.4 Machine model for relaxed HySCharts. 107
5.5 (Relaxed) HySChart for the abstract controller. 110
5.6 Machine model for DiSCharts. 113
5.7 Exemplary behavior of *sample* and *hold*. 119
5.8 DiSChart for the controller. 120
5.9 Intended relation between the exact HySChart, the relaxed HySChart and the DiSChart. 127
5.10 Composition of discrete-time components within HyACharts. 146
5.11 Discrete-time implementation of $RCom$. 148

6.1 HyAChart of the EHC system. 158
6.2 HySChart for the control system *CtrlSys*. 159
6.3 General structure of a closed-loop control system. 159
6.4 Robustness. 162
6.5 General stability. 164
6.6 State-based stability of points. 167
6.7 Global attraction of A. 170

6.8 Classification of properties. 174
6.9 Commuting diagram for L. 182
6.10 Relevant intervals and constants for the chassis level. 189
6.11 An example configuration for N=7 and M=7. 191

Glossary of Symbols

The operators and connectors in hierarchic graphs are defined in Section 3.3 and not repeated here.

General Symbols

$\mathbb{B}$	booleans
$\mathbb{N}$	natural numbers including zero
$\mathbb{R}$	real numbers
$\mathbb{R}_+$	non-negative real numbers
Int	set of all left closed and right open intervals over $\mathbb{R}_+$
$\mathcal{I}$	input space
$\mathcal{O}$	output space
$\mathcal{S}$	state space, data-state space of a HySChart or DiSChart
$\mathcal{C}$	set of causes
$\mathcal{E}$	set of effects
$\wp(X)$	set of all subsets of X, including the empty set
$\mathcal{P}(X)$	set of all *nonempty* subsets of X
$\mathfrak{O}_X$	topology on space X
$(X, \mathfrak{O}_X)$	topological space
π_X	Cartesian projection on space Y
$X^{\mathbb{N}}$	set of functions from $\mathbb{N}$ to X

$X^{\mathbb{R}_+}$	set of functions from $\mathbb{R}_+$ to X
$X^{\mathbb{R}_{p+}}$	set of piecewise smooth, piecewise Lipschitz continuous functions from $\mathbb{R}_+$ to X
$X^{\mathbb{R}_{s+}}$	set of smooth functions from $\mathbb{R}_+$ to X
$r^\dagger$	time extension of relation r
φ^t	right shift of stream φ by t, $\varphi^t(x) = \varphi(x-t)$
$r\|_\delta$	restriction of stream or set of streams r to time interval δ
$A \preceq B$	A is a refinement of B

Symbols used in HyCharts and DiCharts

$n \cdot \mathcal{S}$	program-state space of a HySChart or DiSChart
Com	discrete part of a HySChart
Ana	analog part of a HySChart
Lim	infinitesimal delay
Out	output projection of a HySChart
$RCom$	discrete part of a relaxed HySChart
$RAna$	analog part of a relaxed HySChart
$ROut$	output projection of a relaxed HySChart
R_{int}	output relaxation of a relaxed HySChart
St	state-based semantics of a HySChart (exact or relaxed)
$DCom$	discrete part of a DiSChart
$DAna$	analog part of a DiSChart
Δ	discrete delay by one time unit
$DOut$	output projection of a DiSChart
DSt	state-based semantics of a DiSChart

Act	activity
$da(.)$	adaptation of activities to discontinuities in the input and to transitions by the discrete part
$c.t$	time stamp associated with discrete or hybrid channel c
$c.val$	value associated with channel c
$cs(D)$	streams with correct time stamp information for a discrete or hybrid channel
$x^{\backprime}$	in action guards: current input on channel x
$mes(c)$	minimum event separation on discrete or hybrid channel c
$Li(c)$	Lipschitz constant constraining the continuous periods of evolution on hybrid or continuous channel c
x'	in action bodies: next value of variable x
$e?$	macro denoting a predicate which tests the presence of event e
$e!$	macro denoting a predicate which sends event e
$\varepsilon_{dis}.x$	permitted discretization error for controlled variable x
$\varepsilon_{int}.x$	output relaxation constant for output variable x, limits the permitted intersample error for x
now	private variable denoting a HySChart's or DiSChart's local clock
rs	private variable of a DiSChart signaling initialization and the taking of transitions
$sample_{\mathcal{I},T}$	sampling of a dense stream of type $\mathcal{I}$ with rate T
$hold_{\mathcal{O},T}$	zero-order hold of a discrete-time stream of type $\mathcal{O}$ with rate T
$\multimap\!\!\Rightarrow$	discrete step relation of a HySChart
$\sim\!\!\circ\!\!\Rightarrow$	analog step relation of a HySChart
$\boxminus\!\!\Rightarrow$	logic step relation of a DiSChart
$\sim\!\!\boxminus\!\!\Rightarrow$	control step relation of a DiSChart

Index

$A^{\mathcal{T}}$, 50
$M^{\mathbb{R}_{p+}}$, 41
$M^{\mathbb{R}_{s+}}$, 40
$a|_\delta$, 37
$c.t$, 63
$c.val$, 63
$\rightsquigarrow$, 211
$\dashrightarrow$, 211
Δ, 112
$\rightsquigarrow$, 216
$\dashrightarrow$, 216
$e?$, 69
$e!$, 69
ε_{dis}, 105
ε_{int}, 105
π_X, 124
$\preceq$, 61
φ^t, 41
Ana, 41, 77
Cmp, 42, 53
Cmp_t, 205
Com, 39, 65, 73
Control, 74
DAna, 114
DCom, 114
Filter, 81
Li, 63
Lim, 38
Out, 43
RAna, 106
RCom, 107
St, 42, 108, 205
St_t, 205
$\mathcal{S}$, 38
$\mathcal{S}_{aux}$, 115
$\mathcal{S}_{re}$, 115
$\mathcal{T}$, 49
$cs(D)$, 123
$hold_{\mathcal{O},T}$, 119
$merge_T$, 148
mes, 63
now, 39
  activity, 79
  computation law, 115
$n \cdot \mathcal{S}$, 39, 54
rs, 114, 117
$sample_{\mathcal{I},T}$, 119

abstraction, 88, 109, 180, 182
action, 68
  body, 68
  entry, 66
  exit, 66
  guard, 68
  guard syntax, 101
  preemptive, 70
activity, 40, 77
  composed, 58, 79
  discrete-time, 114
  primitive, 61, 79
  relaxed, 106
  syntax, 77
additive model, 53
admissibility, 213
analog part, 40, 77
  relaxed, 105
analog step relation, 211
analog system, 2
assumption/commitment, 144
attraction, 168, 183
  of points, 169

of trajectories, 168
state-based, 170
AutoFocus, 19, 23, 92

behavior
black box, 160
state-based, 160
white box, 160
block diagram, 12, 13, 18, 82
bond graphs, 13
Büchi automaton, 166

Catalysis, 11
Cauchy sequence, 232
channel
continuous, 62
discrete, 62
hybrid, 62
chaos, 168
Charon, 85
closed set, 228
compositional, 61
computation law, 114
computation unit, 66
computational induction, 213
continuous function, 229
continuous-time system, 2
control law, 3
control state, 54, 73
control step relation, 216
control-flow graph, 53
controllability, 173
controlled space, 134
controlled variables, 38
controller, 10
synthesis, 12
current input, 69

data-flow graph, 48
data-state space, 38, 54
delay element, 82
dense communication history, 41, 50
dense stream, 41, 50
development process
conventional, 16
integrated, 19
DiACharts, 145
digital signal processing, 98, 154
digital system, 2
DiSChart, 112
denotational semantics, 118
machine model, 112
operational semantics, 217
discontinuity adaptation, 41, 116
discrete part, 39, 65
discrete step relation, 211
discrete-time communication history, 50
discrete-time stream, 50
discrete-time system, 2
discretization error, 97, 105
disjoint sum, 54
divergence closed, 206, 234
divergence closure, 206

entry point, 53, 66
environment, 3
Esterel, 26
Euler backward, 136
event, 69
evolution constraints, 123
exit point, 53, 66

factor space, 92, 124
feedback, 46
additive, 56
multiplicative, 51

Galois connection, 180

HDFG, 26
hierarchic graph, 45
homeomorphism, 181
HyAChart, 62
hybrid automaton, 83
hybrid data-flow graphs, 26
hybrid I/O automaton, 84
hybrid module, 83

hybrid step relation, 211
  discrete-time, 216
hybrid system, 1
  control-centered, 177
  process-centered, 178
HyCharts, 31
HySC, 22
HySChart, 64
  denotational semantics, 42, 53
  operational semantics, 212
HyTech, 23

identification, 47
  additive, 57
  multiplicative, 52
identity, 47
  additive, 57
  multiplicative, 52
  time-extended additive, 61
idle, 72
input space, 39
input/output relation, 50
intersample error, 97, 105
invariant, 68, 71, 167, 172
  exact, 72, 101, 102
  relaxed, 102

latched state, 69
latched variables, 38
Liapunov function, 178
linear, time invariant system, 11
Lipschitz continuous, 40, 231
liveness, 171, 172
logic step relation, 216
Lustre, 26

machine model
  discrete-time, 112
  hybrid, 37
  relaxed, 107
Masaccio, 85
MaSiEd, 23, 85
MATLAB/Simulink/StateFlow, 18, 84
metric space, 231
  complete, 232
  of streams, 232
minimum event separation, 63
MSC, 22
multiplicative model, 48

neighborhood, 228
next state, 69

observability, 173
open set, 228
optimal, 163
order, 227
  complete partial, 227
  dense, 227
  total, 227
  well-founded, 227
output relaxation, 105, 106
output space, 39

persistence, 171
PID, 12
piecewise
  Lipschitz continuous, 41
  smooth, 41
plant, 3, 10
pole assignment, 12
preemption, 70
prefix monotonous sequence, 206
program-state space, 39, 54
property
  existential, 172
  universal, 172
Ptolemy, 25

quadratic optimal control, 139

ramification, 47
  additive, 58
  multiplicative, 53
receptive, 44
refinement, 61, 87, 176
  architecture, 88
  behavior, 90

sample-and-hold, 125
resolvable, 42, 80
Ricatti, 10
robust, 162
ROOM, 23

safety, 168, 172
sample-and-hold-refinement, 125
satisfy an activity, 106
SCR, 27, 152
SDL, 26
self-stabilizing algorithm, 190
sequence chart, 22
sequential composition, 46
additive, 55
multiplicative, 50
time-extended additive, 60
Shannon, 11
Signal, 26
simulation
U, 88
U^{-1}, 88
smooth, 40
stability, 164
asymptotic, 171
of points, 166
of trajectories, 165
Poisson, 166
state-based, 167, 178
stability constraints, 139
state-based semantics, 42
state-based system description, 160
state-space representation, 12
state-transition logic, 3
Statecharts, 33
supernode, 130

time guarded, 37, 160
time invariant, 161
time models, 49
time restriction, 37
time stamp, 63
latched, 39
time synchronous, 50
time-divergent prefix monotonous sequence, 206
time-extended additive model, 58
topological conjugacy, 182
topological space, 228
topology
discrete, 228
interval, 229
left, 229
on the real line, 231
subspace, 230
Tychonoff, 230
trace induction, 212
transition
delayed, 102
eager, 72
transposition, 47
additive, 58
multiplicative, 53
trapezoidal method, 136

UML, 23
unconstrained space, 134
uniform monotonicity, 185

V-model, 11
visual attachment, 46
additive, 56
multiplicative, 51
time-extended additive, 60

wait entry point, 66
wait exit point, 67

zero-order hold, 97, 119