TUM

INSTITUT FÜR INFORMATIK

Modular and Visual Specification of Hybrid Systems
– An Introduction to HyCharts –

Radu Grosu and Thomas Stauner

TUM-I9801
Dezember 98

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-12-I9801-80/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Modular and Visual Specification of Hybrid Systems

– An Introduction to HyCharts –

Radu Grosu and Thomas Stauner[1]
Institut für Informatik, Technische Universität München
D-80290 München, Germany
`http://www4.informatik.tu-muenchen.de/`
Email: {grosu,stauner}@informatik.tu-muenchen.de

[1]The second author was supported with funds of the Deutsche Forschungsgemeinschaft under reference number Br 887/9-1 within the priority program *Design and design methodology of embedded systems.*

Abstract

Visual description techniques are particularly important for the design of hybrid systems because specifications of such systems usually have to be discussed between engineers from a number of different disciplines. Modularity is vital for hybrid systems not only because it allows to handle large systems, but also because hybrid systems are naturally decomposed into the system itself and its environment.

Based on two different interpretations for hierarchic graphs and on a clear hybrid computation model, we develop *HyCharts*. HyCharts consist of two modular visual formalisms, one for the specification of the architecture and one for the specification of the behavior of hybrid systems. The operators on hierarchic graphs enable us to give a surprisingly simple denotational semantics for many concepts known from statechart-like formalisms. Due to a very general composition operator, HyCharts can easily be composed with description techniques from other engineering disciplines. Such heterogeneous system specifications seem to be particularly appropriate for hybrid systems because of their interdisciplinary character.

Contents

Chapter 1

Introduction

Hybrid systems have been a very active area of research over the past few years and a number of specification techniques have been developed for such systems. While they are all well suited for closed systems, the search for hybrid description techniques for open systems is relatively new.

For open systems – as well as for any large system – 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 collection of operators on hierarchic graphs as our tool-set, we follow the ideas in [GSB98] and define a simple and powerful computation model for hybrid systems. Based on this model we introduce HyCharts, which consist of two different interpretations of hierarchic graphs. Under one interpretation the graphs are called *HySCharts* under the other one they are called *HyACharts*. HySCharts are a visual representation of hybrid, hierarchic *state transition* diagrams. HyACharts are a visual representation of hybrid *data-flow* graphs (or architecture graphs) and allow the designer to compose hybrid 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 semantics in terms of dense input/output relations. This includes differential equations. Dense input/output relations are a relational extension of hybrid Focus [MS97, Bro97].

1.1 An Example

The following example illustrates the kinds of systems we target at. It will be used throughout the paper to demonstrate the use of HyCharts.

Example 1 (An electronic height control system, EHC) The purpose of this system, 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*[1] which operates the compressor and the escape valve and resets the filter when necessary. A further sensor, *inBend*, tells the controller whether the car is going through a curve.

The diagram in Figure 1.1, 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 further in this paper. Instead we concentrate on the open system consisting of the filter, the controller and a delay element that ensures that the feed-back is well-defined. The escape valve and the compressor are modeled within the controller.

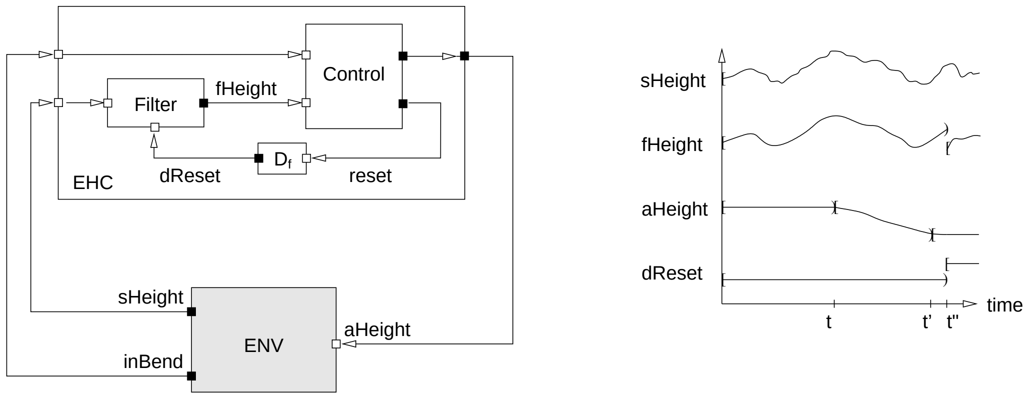

Figure 1.1: The EHC: Architecture and a typical evolution.

Diagrams like the one in Figure 1.1, 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 1.1, right, by periods where the values on the channels change smoothly and by time instances at which there are discontinuities. In our approach the smooth periods result from the analog parts of the components. The discontinuities are caused by their combinational (or discrete) parts.

[1]Note that periodical sampling can be avoided in a hybrid model. However, it was used in the BMW implementation and it allows us to expose various features of our formalism, like entry/exit actions and timeouts.

We specify the behavior of both the combinational and the analog part of a component within a single HySChart, i.e., by a hybrid, hierarchic state transition diagram, with nodes marked by activities and transitions marked by actions. The transitions define the discontinuities, i.e., the instantaneous actions performed by the combinational part. The activities define the smooth periods, i.e., the time consuming behavior of the analog part while the combinational part is idle. As

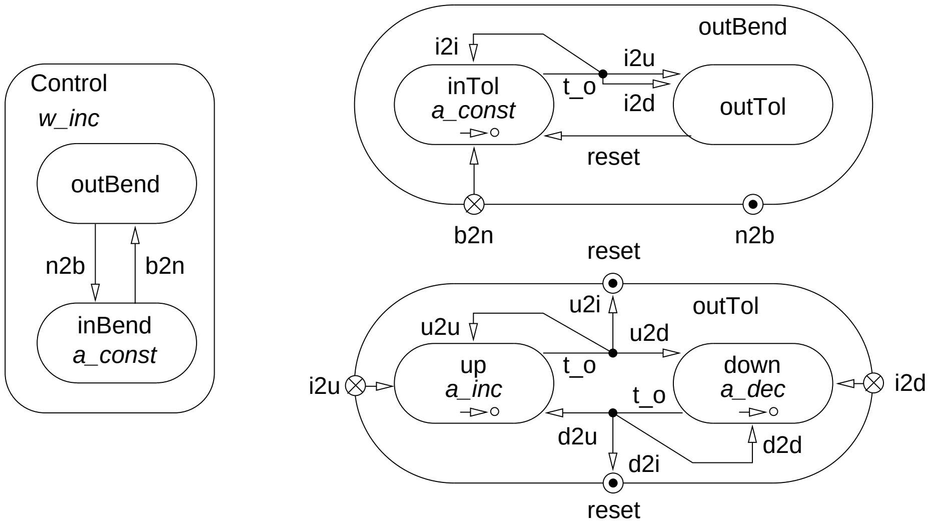

Figure 1.2: The EHC's $Control$ component.

an example, Figure 1.2 shows the HySChart for the EHC's $Control$ component. It consists of three hierarchic levels. Figure 1.2, left, depicts the highest level. Figure 1.2, top right, refines the state *outBend* and Figure 1.2, bottom right, further refines the state *outTol*. The states, transitions and activities (written in italics in the figure) are explained in Chapter 5. □

1.2 Related Work

The basic motivation for this work were experiences we obtained when modeling the EHC case study outlined above with hybrid automata [SMF97]. A basic result of the case study was that the lack of modularity of hybrid automata complicates specification and analysis. Furthermore, the lack of hierarchic states turned out to be inconvenient for specification. In this work, we develop 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 [ACH$^+$95], HyCharts are fully modular and suitable for open systems.

The rather new hybrid modules from Alur and Henzinger [AH97] are modular, but their utility suffers from the fact that it is not obvious how to model feedback loops. For theoretical reasons, loops pose a problem in our approach, too. We solve it by explicitly allowing feedback loops, as long as they introduce a delay. Demanding a delay is not unrealistic, as signals cannot be transmitted at infinite speed.

Another modular model, hybrid I/O automata, is presented in [LSVW96]. While this model is promising from the theoretical point of view, we think it has some deficits in practice. In particular, there is no graphical representation for hybrid I/O automata yet, there is no hierarchy concept for them and finally, there is no visual formalism for the specification of the architecture of a composed system. The same applies for the hybrid modules mentioned above. From the systems engineering point of view our approach is therefore more convenient.

A first approach towards a hybrid version of statecharts can be found in [KP92]. The operational semantics given there, however, does not allow inter-level 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 get 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 makes it difficult to combine the above models with models for continuous systems, as they evolved in the engineering disciplines, difficult. A decision must be made that determines which value of the variable is to be "exported", i.e. visible to the outside world at a physical time instant. 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 has exactly one value at each time instant, the variable evaluation is a function of time.

Commercial products for the design of embedded systems, like StateFlow [TMI98], take a different approach to specifying hybrid systems. In this approach the system needs to be partitioned into purely discrete and purely continuous components before specification can begin. While this method may be appropriate for many systems we think it enforces a too early partitioning into hardware and software components and is highly inconvenient for specifying components that are hybrid themselves, like some environment models, which, for example, contain phase transitions. A formal model that uses a specification approach similar to StateFlow can be found in [EH96]. Interestingly there are some parallels between our hybrid machine model (Chapter 3) and the model presented there.

To end this journey through the literature we want to mention that HyCharts look largely similar to the description techniques used in the software engineering method for real-time object-oriented systems ROOM [SGW94] and may therefore be seen as a hybrid extension of them.

1.3 Overview

The rest of the paper is organized as follows. In Chapter 2 we present two abstract interpretations of hierarchic graphs. These interpretations provide the infrastructure for defining a surprisingly simple denotational semantics for the key concepts of statecharts [Har87] offered in HyCharts, like hierarchy and preemption. They also are the foundation for the denotational semantics of our hybrid computation model, which is introduced in Chapter 3. Following the ideas developed in this model, HyCharts are defined in Chapters 4 and 5 as a multiplicative and an additive interpretation of hierarchic graphs, respectively. Both diagram kinds are introduced in an intuitive way by using the example above. In Chapter 6 we briefly discuss how other techniques for component specification can be integrated into our approach and relate HySCharts to timed automata. Furthermore, we discuss the HyChart specification of the filter component from our example system and its importance for hybrid modeling. Finally, in Chapter 7 we summarize our results.

Chapter 2

Hierarchic Graphs

This chapter first introduces an algebra of hierarchic graphs (Section 2.1). Then two models, an *additive* and a *multiplicative* model, for this algebra are given. The additive model interprets the operators in a way that results in control-flow graphs (Section 2.2), the multiplicative model interprets them in a way that yields data-flow graphs (Section 2.3).

2.1 Syntax

A *hierarchic graph* consists of a set of *nodes* connected by a set of *arcs*. For each node, the incoming and the outgoing arcs define the node's *interface*, i.e. its *type*. Let A and B be the input and the output interfaces of a node n. Then the corresponding textual notation for n is $n : A \to B$ (Fig. 2.1). Interpreting A and B as sets (or types) n may be regarded as a mapping from elements of A into elements of B.

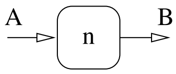

Figure 2.1: A node $n : A \to B$.

2.1.1 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 2.2.

Sequential composition. One basic way to connect two nodes is by *sequential composition,* i.e., as shown in Figure 2.2, left, by connecting the output of one

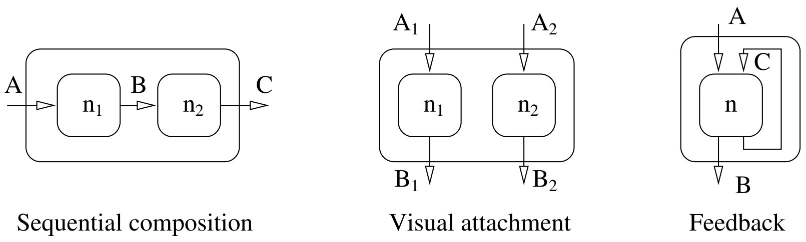

Figure 2.2: The composition operators.

node to the input of the other node, if they have the same type. Textually we denote this operator by the semicolon ;. Given $n_1 : A \to B$ and $n_2 : B \to C$ we define $n_1; n_2$ to be of type $n_1; n_2 : A \to C$.

Regarding the nodes as computation units, Figure 2.2, 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 arrows are put one near the other, as shown in Figure 2.2, middle. To obtain a textual representation for visual attachment, we need an attachment operator both on arrows and on nodes. We denote this operator by $\star$. Given two arrows A and B their visual attachment is expressed by $A \star B$. Given two nodes $n_1 : A_1 \to B_1$ and $n_2 : A_2 \to B_2$ their visual attachment is expressed as $n_1 \star n_2 : A_1 \star A_2 \to B_1 \star B_2$. 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 deal with *hiding* it is convenient to explicitly introduce an arrow E denoting the *absence* of any information. This arrow is neutral for attachment, i.e. $A \star E = E \star A = A$, because visually attaching nothing does not change the original information.

Feedback. Sequential composition allows us to connect the nodes of a graph in a causal way. However, using only sequential composition to connect nodes is not expressive enough because it cannot deal with loops or with bidirectional communication. We therefore introduce a *feedback operator*, as shown in Figure 2.2, 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 $n : A \star C \to B \star C$ we define $n \uparrow^{C}_{A,B}: A \to B$. Similar to sequential composition and visual attachment, feedback also introduces a containment relationship. We say that n and the feedback arrow are contained in $n \uparrow^{C}_{A,B}$.

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

2.1.2 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 2.3.

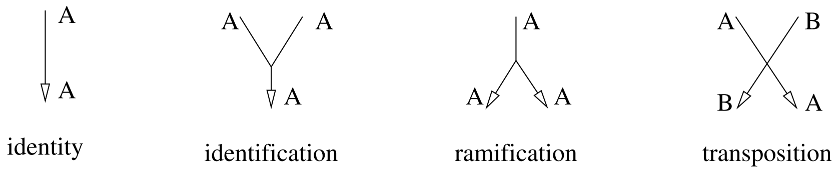

Figure 2.3: The connectors.

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

Identification. The *identification* connector $\vee_A^k$ joins k inputs together. Its type is $A^k \to A$, 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 arrow. In Figure 2.3 the binary case is depicted.

Ramification. The *ramification* connector $\wedge_k^A$ copies the input information on k outputs. Hence $\wedge_k^A$ has type $A \to A^k$. Figure 2.3 shows the binary case.

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

To be a precise formalization of our intuitive understanding of graphs, the above abstract operators and connectors have to satisfy a set of laws, which intuitively express our visual understanding of graphs. These laws correspond to *strict, symmetric, monoidal categories with feedback* and *bimonoid objects*, see e.g. [Ste94]. [GSB98] shows that the *additive* and the *multiplicative interpretations* of the operators and connectors are particularly relevant for computer science.

2.1.3 An Example

As an example for a hierarchic graph and its corresponding textual respresentation we consider the graph in Figure 2.4, left. Using the above basic operators and connectors it defines a *derived* composition operator, the *symmetric feedback.* If $n_1 : A_1 \star A \to B_1 \star B$ and $n_2 : B \star A_2 \to A \star B_2$ then $n_1 \circledast n_2$ has type $A_1 \star A_2 \to B_1 \star B_2$. Its simplified visual representation is given in Figure 2.4, right. The textual respresentation corresponds one to one to the visual representation in Figure 2.4, left:

$$n_1 \circledast n_2 = (((\mathsf{I}_{A_1} \star {}^{A_2}\mathsf{X}^{A \star B});(n_1 \star n_2);(\mathsf{I}_{B_1} \star {}^{B \star A}\mathsf{X}^{B_2});(\mathsf{I}_{B_1 \star B_2} \star {}^{B}\mathsf{X}^{A}))\uparrow_\star^B)\uparrow_\star^A$$

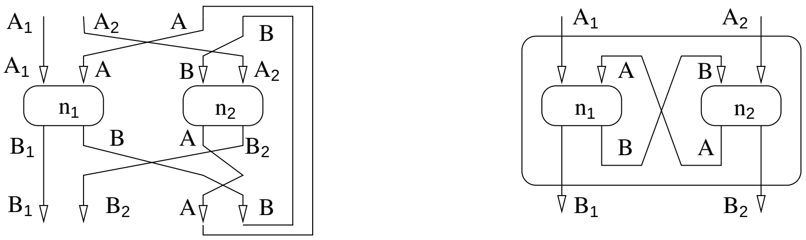

Figure 2.4: The symmetric feedback.

2.2 The Additive Model

The *additive model* is a model for *hierarchic control-flow graphs*. The intuition behind these graphs is as follows. At any moment of time, the control resides in *exactly one* node. The node receives the control on one of its *entry points* and gives the control back on one of its *exit points*. Entry and exit point 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 disjointness 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.

2.2.1 Arrows

The *control* can be understood as a pair (k, s) consisting of the *control-state (or program counter)* $k \in \mathbb{N}$ and the *data-state* $s \in \mathcal{S}$. As a consequence, we consider given a set $\mathcal{S}$, the *data-state space*. Each primitive arrow in a flow-graph is *interpreted* as this set. The *control-state space* $\mathbb{N}$ is not an arrow, but it appears in the definition of visual attachment. We take the *empty set* as the interpretation of the *neutral arrow* E. Given the data-state space $\mathcal{S}$ we define the *program-state space* $n{\cdot}\mathcal{S}$ as follows:

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

Now the *disjoint sum* of program-state spaces is defined by the following equation: $m{\cdot}\mathcal{S} + n{\cdot}\mathcal{S} = (m+n){\cdot}\mathcal{S}$. 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:

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

We interpret the *visual attachment* operator in control-flow graphs by the *disjoint-sum* operator. 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 merely refer to the program-state as the state.

2.2.2 Nodes and Operators

A node $n : A \to B$ 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 input*, the *current control* and the *next control*. Upon receiving the current control state, it determines the next control state, depending on 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 inputs and produce outputs. The output space simply is a projection of the data-state space. We write $\mathcal{I}$ to denote the input space. The definition of the operators below ensures that all nodes receive the same input. Therefore, by convention no arrow is drawn to denote the external input $\mathcal{I}$ 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 $x \cdot \mathcal{S}$ and $x_i \cdot \mathcal{S}$ over data-state space $\mathcal{S}$ by X and X_i for $x \in \{a, b, c\}$.

The Node Operators

Sequential composition. The 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:

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

Additive composition. The additive composition 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 in statecharts, a new node $n_1 + n_2$, such that control resides either in n_1 or in n_2. Note that the interface of the sum reflects this fact.

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

The visual notation of $n_1 + n_2$ is given in Figure 2.5, left. The meaning of $(n_1 + n_2)(x, l.a)$ is intuitively shown in Figure 2.5, right. Receiving the tuple $(x, l.a)$, the sum uses the control information l to "demultiplex" the input and select the corresponding relation n_1; this relation is then applied to (x, a) to

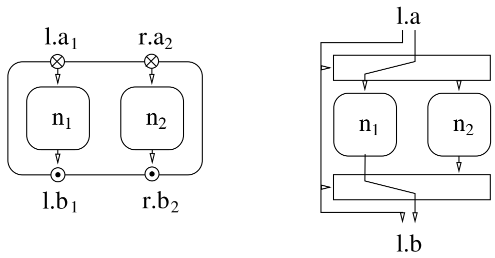

Figure 2.5: The additive interpretation.

obtain the next state b; finally, the output of the relation is "multiplexed" to $l.b$.

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 control is received on A and it is either given directly on B or after an arbitrary number of times in which it loops along C. Formally:

$$\begin{aligned} n{\uparrow}^C_+ \;\; &\subseteq (\mathcal{I} \times A) \;\times\; B \\ n{\uparrow}^C_+ \;\; &= n_{l,l} \;\cup\; n_{l,r} \,;\, n^*_{r,r} \,;\, n_{r,l} \end{aligned}$$

where n^* is the arbitrary but finite iteration of n and $n_{i,j}$ is 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.

The Connectors

Identity. The identity I_A is defined as expected:

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

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

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

where $k\,A = k\,(a \cdot \mathcal{S}) = (k \cdot a) \cdot \mathcal{S}$.

Additive ramification. The additive ramification ${}_A{\bullet\!\!<}_k$ gives the control on any of its exit points:

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

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

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

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

2.3 The Multiplicative Model

The *multiplicative model* is a model for *hierarchic data-flow graphs*. The intuition behind these graphs is as follows. At any moment of time, *all* nodes of the graph are active and computing the output data based on the 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 intended parallelism of nodes, input/output channels and branches of the connectors is obtained by interpreting the *visual attachment* $\star$ *multiplicatively* by the *product* $\times$ and by defining the other operators and connectors consistently with this interpretation.

2.3.1 Arrows

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

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

Given 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:

$$
\begin{aligned}
A \times \{()\} \;&= \{()\} \times A = A \\
A \times B \;&= A_1 \times \ldots \times A_m \times B_1 \ldots \times B_n
\end{aligned}
$$

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

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

The projections 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. The unique existence of projections is characteristic for data-flow graphs.

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*. In a hybrid system this flow may be continuous (think of analog devices), so we assume that time increases continuously, i.e. it is *dense*, and take the non-negative real numbers $\mathbb{R}_+$ as abstract time axis. In this case, the data exchanged along a channel with type A over time defines a mapping $a \in A^{\mathbb{R}_+}$. Motivated by our hybrid computation model (Chapter 3) we impose some restrictions on this mapping in the next chapter. We call such a restricted mapping a *dense communication history* and its corresponding type a *dense communication history type*. The latter ones are used to interpret the *primitive arrows* of data-flow graphs.

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 (and its associated prefix-ordering) of a component's interface $(A_1 \times \ldots \times A_m)^{\mathbb{R}_+}$ is equal to the product $A_1^{\mathbb{R}_+} \times \ldots \times A_m^{\mathbb{R}_+}$ of the histories of its channels.

2.3.2 Nodes and Operators

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. The relation must be total in the input histories. We assume that the relations are defined such that the data occurring in the histories of the output channels at time t only depends on the input history received up to (and including) t. Formally, for all a_1, a_2 and t:

$$a_1{\downarrow}_{[0,t]} = a_2{\downarrow}_{[0,t]} \quad \Rightarrow \quad n(a_1){\downarrow}_{[0,t]} = n(a_2){\downarrow}_{[0,t]}$$

where by $a{\downarrow}_\delta$ we denote the restriction of a to the time interval δ. Clearly, each realizable component behaves in this way. We call these relations *time guarded.*

They *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). Note, however, that the names and symbols are syntactic entities whereas the relations and relational operators are semantic entities.

The Node Operators

Sequential composition. The 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:

$$n_1 \subseteq A^{\mathbb{R}+} \times B^{\mathbb{R}+}, \quad n_2 \subseteq B^{\mathbb{R}+} \times C^{\mathbb{R}+}$$

we define their sequential composition $n_1 \,;\, n_2$ as follows:

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

Parallel composition. As in statecharts, the parallel composition of two components yields a new component such that *both* constituents are active simultaneously, i.e. each constituent has its own control, described for example by a control-flow graph. The interface of the product has to reflect this fact. Given two relations

$$n_1 \subseteq {A_1}^{\mathbb{R}+} \times {B_1}^{\mathbb{R}+}, \quad n_2 \subseteq {A_2}^{\mathbb{R}+} \times {B_2}^{\mathbb{R}+}$$

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

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

The visual notation for $n_1 \times n_2$ is given in Figure 2.6.

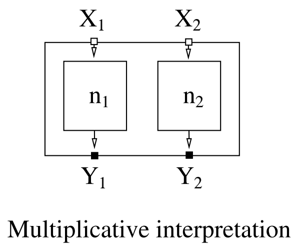

Figure 2.6: The multiplicative interpretation.

Feedback. The multiplicative feedback allows the passing of the output of a component back to its input. In the next chapter we will use this construct to add the memory to our components. Given a relation:

$$n \subseteq (A^{\mathbb{R}+} \times C^{\mathbb{R}+}) \;\times\; (B^{\mathbb{R}+} \times C^{\mathbb{R}+})$$

we define the new relation $n{\uparrow}^C_\times$ as below:

$$\begin{aligned} n{\uparrow}^C_\times &\subseteq A^{\mathbb{R}+} \;\times\; B^{\mathbb{R}+} \\ n{\uparrow}^C_\times &= \{(a,b) \mid \exists c.\ (b,c) \in n(a,c)\} \end{aligned}$$

$n{\uparrow}^C_\times$ is time guarded and guaranteed to be total in the input channel histories $A^{\mathbb{R}+}$ if n is time guarded and its output on channel C up to time $t+\delta$ is completely determined by its input up to time t on input channel C and by the input on the other input channels up to time $t+\delta$ [Bro97]. I.e. its output on C reacts with a delay $\delta > 0$ to input channel C. We also say that n is *strongly* time guarded on feedback channel C.

The Connectors

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

$$\mathsf{I}_A \;\subseteq A^{\mathbb{R}+} \times A^{\mathbb{R}+}, \qquad \mathsf{I}_A \;= \{(a,a) \mid a \in A^{\mathbb{R}+}\}$$

Multiplicative identification. The identification connectors $\curlyvee^k_A : A^k \to A$ are interpreted by the multiplicative identification relations $\curlyvee^k_A$. They allow to identify k copies of elements $a \in A^{\mathbb{R}+}$:

$$\curlyvee^k_A \;\subseteq (A^k)^{\mathbb{R}+} \times A^{\mathbb{R}+}, \qquad \curlyvee^k_A \;= \{(a^k, a) \mid 0 < k \;\wedge\; a \in A^{\mathbb{R}+}\}$$

Note that $(A^k)^{\mathbb{R}+} = (A^{\mathbb{R}+})^k$, as we are in a time synchronous setting.

Multiplicative ramification. The ramification connectors $\curlywedge^A_k : A \to A^k$ are interpreted by the multiplicative ramification relations $\curlywedge^A_k$. They allow to make k copies of the input a:

$$\curlywedge^A_k \;\subseteq A^{\mathbb{R}+} \times (A^k)^{\mathbb{R}+}, \qquad \curlywedge^A_k \;= \{(a, a^k) \mid 0 < k \;\wedge\; a \in A^{\mathbb{R}+}\}$$

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

$$\begin{aligned} {}^A\mathsf{X}^B &\subseteq (A^{\mathbb{R}+} \times B^{\mathbb{R}+}) \times (B^{\mathbb{R}+} \times A^{\mathbb{R}+}), \\ {}^A\mathsf{X}^B &= \{((a,b),(b,a)) \mid (a,b) \in A^{\mathbb{R}+} \times B^{\mathbb{R}+}\} \end{aligned}$$

Chapter 3

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.

3.1 General Idea

We model a *hybrid system* by a network of autonomous *components* that communicate in a *time synchronous* way. Time synchrony is achieved by letting time flow uniformly for all components.

Each component is modeled by a *hybrid machine*, as shown in Figure 3.1, left. This machine consists of five parts: a *combinational* (or discrete) part (Com), an *analog* (or continuous) part (Ana), a *feedback* loop, an infinitesimal delay (Lim_s), and a projection (Out). The feedback models the *state* of the machine. Together

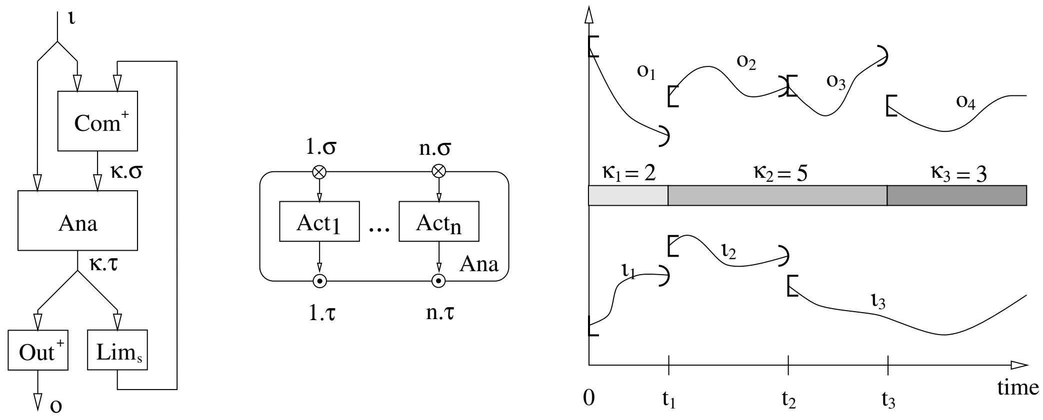

Figure 3.1: The hybrid-machine computation model.

with Lim_s it allows the component to remember at each moment of time t the input received and the output produced "just before" t.

The combinational part 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 combinational part passes the fed back state without modification, we say that it is *idle*. The combinational 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, provided the input does not change discretely during the interval.

The analog part describes the input/output behavior of the component whenever the combinational part is idle. Hence, it adds to the component the temporal dimension. It may select a new activity whenever there is a discrete change in the input it receives from the environment or the combinational part.

Example 2 Figure 3.1, right, shows the exemplary behavior of a component. The shaded boxes κ_i indicate the time periods where the combinational part idles in node i. At time t_1 the discrete move of the environment triggers a discrete move of the combinational part. According to the new next state received from the combinational part, the analog part selects a new activity. The activity's start value at time t_1 is as determined by the combinational part. At time t_2 there is a discrete move of the environment, but the combinational part remains idle. The analog part chooses a new trajectory for the variables whose start value is the analog part's output just before t_2, because this is what it receives from the combinational part at time t_2. Thus, the output has a higher order discontinuity here. At time t_3 the environment does not perform a discrete move, but the combinational part does, e.g. because some threshold is reached. Again the analog part selects a new activity, which begins with the start value determined by the combinational part. During the intervals $(0, t_1), (t_1, t_3)$ and (t_3, ∞) the combinational part is idle. □

Please note the structural similarity of our hybrid machine and discrete controllers of continuous systems in Control Theory. There we also have a discrete and a continuous part that are interconnected with feedback.

Feedback and state. Since the input received and the output produced may change abruptly at any time t, as shown in Figure 3.1, right, we consider that the state of the component at moment t is the limit $lim_{x \nearrow t} \psi(x)$ of all the outputs $\psi(x)$ produced by the analog part when x approaches t. 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. 3.1, left). Its definition is:

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

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 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 $\mathcal{S}_P$ for $\prod_{v \in P} \sigma_v$ and $\mathcal{S}_O$ for $\prod_{v \in O} \sigma_v$. Clearly, $\mathcal{S} = \mathcal{S}_P \times \mathcal{S}_O$. The latched inputs are a subset of P.

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 is given by $\mathcal{I} = \prod_{v \in I} \sigma_v$.

3.2 The Combinational Part

The combinational 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 $n \cdot \mathcal{S}$ is the *program-state space* (see Section 2.2.1) and $\mathcal{P}(X) = \{Y \subseteq X \mid Y \neq \{\}\}$. The n is the number of leaf nodes in the hierarchic graph that defines Com (see Section 5.1).[1] The computation of Com takes no time.

An important property of the relation defining the combinational 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. Furthermore, we want that the combinational part passes the next state to the analog part only if it (the combination part) cannot proceed further. In other words, if $s' \in Com(i, s)$ is the next state, then $Com(i, s') = \{s'\}$, i.e., no new state $s" \neq s'$ can be computed starting in s' with input i. We say that Com is idle for i and s'. Finally, the set $E \subseteq \mathcal{I} \times n \cdot \mathcal{S}$ of inputs and states for which Com is not idle must be topologically closed.[2] Together with the preceding property this guarantees that the extension of Com over time can only make discrete moves at distinct points in time. This fact is needed in the following to ensure that the semantics of a hybrid machine is well-defined.

[1] Technically the output of Com is an element of a disjoint sum of some structure with n summands $\mathcal{S}$. Due to associativity of the disjoint some we abbreviate this as $n \cdot \mathcal{S}$.

[2] As topology we use the Tychonoff topology on $\mathcal{I} \times n \cdot \mathcal{S}$ which is induced by using the discrete topologies on the variable domains different from $\mathbb{R}$ and the Euclidean topology on $\mathbb{R}$ for the variable domains that are equal to $\mathbb{R}$ [Eng89].

3.3 The Analog Part

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

$$Act \quad \in \quad (\mathcal{I} \times \mathcal{S})^{\mathbb{R}_{c+}} \rightarrow \mathcal{P}(\mathcal{S}^{\mathbb{R}_{c+}})$$

For any set M, the set $M^{\mathbb{R}_{c+}}$ stands for the set of functions from the non-negative real numbers $\mathbb{R}_+$ to M that are *continuous* and *piecewise smooth*. We say that a function $f \in \mathbb{R}_+ \rightarrow M$ is piecewise smooth iff every finite interval on the non-negative real line $\mathbb{R}_+$ can be partitioned into *finitely* many left closed and right open intervals such that on each interval 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. A tuple of functions is infinitely smooth iff all its components are. We also call $M^{\mathbb{R}_{c+}}$ the set of *flows* over M. To model analog behavior in a "well behaved" way, activities must be total and time guarded. Furthermore, we demand that the activities do not depend on absolute time (measured from system start) but may be started anytime. Using a relational notation for *Act* this formally means that for all time intervals $[u, v)$ and for all histories $\varphi, \psi \in \mathcal{S}^{\mathbb{R}_{c+}}$ and $\iota \in \mathcal{I}^{\mathbb{R}_{c+}}$:

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

where φ^t is the right shift of stream φ by t, $\varphi^t(x) \stackrel{\text{def}}{=} \varphi^t(x - t)$.

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

$$Ana \quad \in \quad (\mathcal{I} \times n \cdot \mathcal{S})^{\mathbb{R}_+} \rightarrow \mathcal{P}((n \cdot \mathcal{S})^{\mathbb{R}_+})$$

where $n \cdot \mathcal{S}$ is the program-state space, as in the type of *Com*, and for any set M, $M^{\mathbb{R}_+}$ denotes the set of *piecewise smooth* functions $\mathbb{R}_+ \rightarrow M$. Hence, the input and output of the analog part is not necessarily continuous. Instead, finitely many discrete moves by the combinational part and the environment during any finite interval are allowed. In the following we will see that this demands that the combinational part is realizable. We call $M^{\mathbb{R}_+}$ the set of *dense communication histories.*

The relation *Ana* is obtained by pasting together the flows of the activities associated to the nodes where the combinational part *Com* idles. Pasting is realized as shown in Figure 3.1, middle, by extending the sum operation to activities. Given a set of activities $ACT = \{Act_j \mid j \leq n\}$, their sum is defined as below:[3]

$$\begin{aligned} +_{j=1}^{n} Act_j \quad \stackrel{\text{def}}{=} \quad \{\; & (\iota, (\kappa, \sigma), (\kappa, \tau)) \mid \\ & \forall \delta, m.\; \kappa|_\delta = m^\dagger \Rightarrow m \leq n \wedge (\iota, \sigma, \tau)|_\delta \in (Act_m)|_\delta \} \end{aligned}$$

[3]Here we use for convenience the relational notation $Act \subseteq \mathcal{I}^{\mathbb{R}_{c+}} \times \mathcal{S}^{\mathbb{R}_{c+}} \times \mathcal{S}^{\mathbb{R}_{c+}}$.

where δ is a left closed right open interval, $m^\dagger$ is the extension of m to a constant function over δ, $\iota \in \mathcal{I}^{\mathbb{R}_+}$ and $(\kappa, \sigma), (\kappa, \tau) \in (n \cdot \mathcal{S})^{\mathbb{R}_+}$. The tuple (κ, σ) consists of the control-state flow κ which gives at each moment of time the node where the combinational part idles (see Figure 3.1, right) and the data-state flow σ which gives at each moment of time the data-state passed by the combinational part. The tuple (κ, τ) consists of the same control-state flow κ and the data-state flow τ computed by the sum. For each interval δ in which the combinational part idles, the sum uses the control information $\kappa|_\delta$ to demultiplex the input $(\kappa, \sigma)|_\delta$ to the appropriate activity and to multiplex the output $\tau|_\delta$ to $(\kappa, \tau)|_\delta$. Section 5.2 will show how *Ana* is constructed from the activities in a HySChart by using the $+$ operator. As the construction results in a flat structure over $\mathcal{S}$, we need not use injections $l.$ and $r.$ in the definition of $+$, but can directly use the summands' numbers in the n-fold disjoint sum $n \cdot \mathcal{S}$.

Note that the type of *Ana* assures that $(\iota, (\kappa, \sigma))$ is partitioned into pieces, where ι, κ and σ are simultaneously piecewise smooth. The output histories (κ, τ) of *Ana* are again piecewise smooth, by the definition of *Ana*.

As we demand that every activity is total and time guarded, the analog part also is total and time-guarded. Furthermore, for the analog part we demand that it is *resolvable*, which means that it must have a fixed point for every state $s_0 \in n \cdot \mathcal{S}$ and every input stream $i \in \mathcal{I}^{\mathbb{R}_{c+}}$, i.e.,

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

Resolvability of the analog part is needed to prove that the semantics of a hybrid machine is well-defined (see below).

3.4 The Component

Given an initial state s_0, the behavior of the hybrid machine is a relation *Cmp* between its input and output communication histories. Writing the graph in Figure 3.1, middle, as a relational expression with the multiplicative operators results in the denotational semantics of *Cmp*:

$$\begin{array}{lcl} Cmp & \in & n \cdot \mathcal{S} \to \mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_+}) \\ Cmp(s) & = & ((\text{Ʌ}_2 \times \mathsf{I}) \;;\; (\mathsf{I} \times Com^\dagger) \;;\; Ana \;;\; \text{Ʌ}_2 \;;\; (Out^\dagger \times Lim_s)) \uparrow_{\times}^{n \cdot \mathcal{S}} \end{array}$$

where $R^\dagger$ trivially extends the combinational relation R in time, i.e. $R^\dagger(\iota) \stackrel{\text{def}}{=} \{o \mid o(t) \in R(\iota(t))\}$ for any $t \geq 0$. *Out* selects the output variables from the state stream.

By definition, *Cmp* is a *time guarded* relation, because $Com^\dagger$, *Ana*, $Out^\dagger$, Lim_s, I and Ʌ_2 are time guarded. To show that *Cmp* is total we outline the proof for the existence of a fixed point of the above definition for arbitrary starting state and input. As the composed relation under the feedback operator does not introduce a delay $\delta > 0$, the existence of a fixed point is not guaranteed a priori. Instead, it is a consequence of the properties of Com and Ana.

Proof for the existence of a fixed point

First, we prove that some time $t > 0$ passes between two discrete moves by the combinational part or the environment. Above, we demanded that the set $E \subseteq \mathcal{I} \times n \cdot \mathcal{S}$ on which Com is not idle is topologically closed. Therefore, E is also closed with respect to the induced subspace topology on $(\mathcal{I} \times n \cdot \mathcal{S}) \cap range((\iota, \sigma)|_{[t_0,t_1)})$ [Eng89]. Now suppose s' is an output of Com for the current input i and the latched state s at time t_0 (see Figure 3.2). From the restrictions

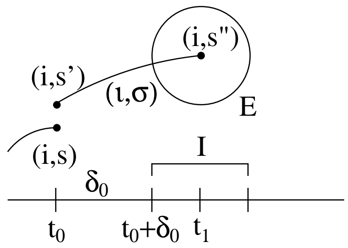

Figure 3.2: Computing the minimal delay.

we imposed on Com we know that it must be idle for s' and the current input, i.e., $(i, s') \notin E$. Com will remain idle as long as its inputs from the environment and the feedback loop are not in E. Hence, we must determine when E can be reached next. As the input stream ι is piecewise infinitely 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. Constructing the inverse image of E for the fixed point of Ana and the input stream up to t_1 yields a set I that is closed w.r.t. $dom((\iota, \sigma)) = [t_0, t_1)$, since the input and the analog part's output are continuous functions up to t_1. As t_0 is not in this set and the set is closed w.r.t. $[t_0, t_1)$, we get that the next discrete move cannot be performed before $t_0 + \delta_0 = min\{min\{I\}, t_1\} > t_0$. ($min\{I\}$ exists, because I is bounded from below and closed.)

On the interval $[t_0, t_0 + \delta_0)$ the fixed point of Ana is a fixed point of Cmp, because Com and Lim_s are the identity there. Applying this argument inductively we get 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 a proper fixed point of Cmp. Otherwise we have a *zeno* execution, the combinational 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 combinational 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.*

3.5 A Note on Semantics

A very important characteristic of our semantic model is its uniform use of the relational framework. Activities and the component itself are both total time guarded relations. This agrees with Abramski's slogan that *processes are relations extended in time.* Moreover, the combinational 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 preceding chapter 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. The time extension of the additive operators leads to activity specifications for the analog part.

Using dense piecewise smooth communication histories as the basis for component specification allows 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].

Chapter 4

System 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 2.1. Writing the graph as the equivalent relational formula and using the multiplicative model to interpret the operators in it directly gives 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 2.3.

The component names in the graph refer to input/output behaviors specified in other HyACharts or with other formalisms (Chapters 5 and 6). The channel names are the input and output variable names used in the specification of the components. The variables' types must be specified separately.

We can now return to the HyAChart of our example system given in the introduction in Figure 1.1, left, and develop its semantics.

Example 3 (HyAChart of the EHC) In Figure 1.1, left, the boolean-valued channel *inBend* signals the controller whether the car is in a curve. The real-valued channel *sHeight* carries the chassis level measured by the sensors. The real-valued channel *fHeight* carries the filtered chassis level. The real-valued channel *aHeight* carries the chassis level as proposed by the actuators, compressor and escape valve, without environmental disturbances. The boolean-valued channels *reset* and *dReset* (delayed reset) transfer the boolean reset signal to the filter. The delay component D_f ensures that the feedback is well-defined (see Section 2.3).

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

$$\begin{array}{lll} Filter & \in & (\mathbb{R} \times \mathbb{B})^{\mathbb{R}_+} \to \mathcal{P}(\mathbb{R}^{\mathbb{R}_+}) \\ Control & \in & (\mathbb{B} \times \mathbb{R})^{\mathbb{R}_+} \to \mathcal{P}((\mathbb{R} \times \mathbb{B})^{\mathbb{R}_+}) \\ D_f & \in & \mathbb{B}^{\mathbb{R}_+} \to \mathcal{P}(\mathbb{B}^{\mathbb{R}_+}) \end{array}$$

The semantics of the whole system EHC is defined as below. It is the relational algebra term corresponding to the HyAChart of Figure 1.1, left.

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

Note that the user only has to draw the HyAChart and to define the types of the channels. □

Chapter 5

Component Specification - HySCharts

A HySChart (Hybrid StateChart) defines the combinational and the analog part of a hybrid machine. The input/output behavior of the resulting component follows from these parts as explained in Chapter 3.

The Graphical Syntax of HySCharts. A HySChart is a hierarchic graph, where each node is of the form depicted in Figure 5.1, left. Each node may have sub-nodes. It is labeled with a node name, which only serves for reference, 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 on the input, the latched state and the next state. They are structured into a *guard* and a *body*. The activity names refer to systems of ordinary differential (in)equations. The specification of actions and activities and their semantics is explained in detail in the following. Transitions from composed nodes express preemption. Except for activities, HySCharts look similar to ROOM-charts [SGW94].

The Semantics of HySCharts. The semantics of a HySChart is divided into a combinational and an analog part. The combinational part follows almost directly from the diagram. The analog part is constructed from the chart with little effort.

In the following we will first explain how the combinational part is derived from a HySChart, then the analog part is covered. We will also show how actions and continuous activities are specified.

5.1 The Combinational Part

A HySChart is a hierarchic graph and therefore constructed from the operators in Section 2.1. As mentioned in Section 2.2, interpreting the graph in the additive model leads to a close correspondence to automata diagrams.

We may view the graph as a network of autonomous *computation units* (the nodes) that communicate with each other over directed *control paths* (the arcs). Due to the additive model, at each time point control resides in only one (primitive) computation unit (Section 2.2).

In order to derive the combinational part from the HySChart we now give a semantics to its nodes, i.e., to its computation units. The semantics for hierarchy and actions follows.

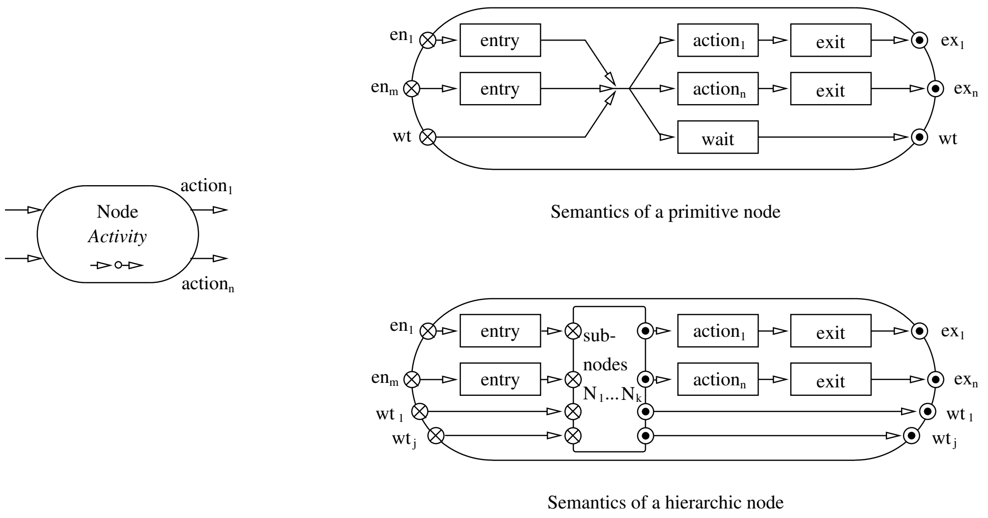

Figure 5.1: Syntax and semantics of a computation unit.

Computation units. Each primitive node of the HySChart represents the graph given in Fig. 5.1, top right. It formally corresponds to the relational expression below:

$$\texttt{CompUnit} \stackrel{\text{def}}{=} (+_{i=1}^{m}\texttt{entry} \;+\; \mathsf{I})\,;\; {}_{m+1}\!\succ\!\bullet \,;\, \bullet\!\prec_{n+1} ;\, ((+_{i=1}^{n}\texttt{action}_i\,;\,\texttt{exit}) + \texttt{wait})$$

According to the additive operators, it has the following intuitive meaning. A computation unit gets the *control* along one of its *entry points* $\texttt{en}_i$ and gives the control back along one of its *exit points* $\texttt{ex}_j$.

After getting control along a regular entry point, i.e., an entry point different from *wait* `wt`, a computation unit first executes its *entry action* `entry`, if one is specified. Then it evaluates a set of *action guards.*[1] 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.

[1] An action $\texttt{action}_k$ consists of a *guard* and a *body.*

If more than one guard is true, then the computation unit *nondeterministically* chooses one of them. Guard `wait` in the diagram stands for the negation of the disjunction of the guards of the actions $\texttt{action}_k$. Hence, if none of the guards is true, then the discrete computation is completed, and the control leaves the combinational part along the designated wait exit point `wt`. The next section shows that the analog part takes advantage of the information about the exit point to determine the activity to be executed and gives control back along the corresponding wait entry point.

Hierarchy. A composed or *hierarchic* node in the HySChart stands for the graph in Figure 5.1, bottom right. A principal difference to primitive nodes is that the entry points are not identified, instead they are connected to the corresponding entry points of the sub-nodes. Similarly, the exit points of the sub-nodes are connected to the corresponding exit points of their enclosing hierarchic node. Furthermore, the hierarchic node has a wait entry and wait exit point for every wait entry/exit point of the sub-nodes. When it receives control on one of them, it is directly passed on to the wait entry point of the corresponding sub-node. Thus, the wait entry point identifies a sub-node. The hierarchic node is left along a wait exit point, if a sub-node is left along its corresponding wait exit point.

Actions. 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. They are 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. We use *left-quoted variables* $v^{\backprime}$ to denote the current input, *right-quoted variables* v' to denote the next data-state and *plain variables* to denote the latched data-state. Moreover, we mention only the changed variables and always assume the necessary equalities stating that the other variables did not change. To simplify notation further, we associate a variable c with each channel c. For example, the action resetting the filter in the EHC example is defined as follows:

$$\mathit{dReset}^{\backprime} \neq \mathit{dReset} \wedge \mathit{dReset}' = \mathit{dReset}^{\backprime} \wedge \mathit{fHeight}' = 0$$

It says that each time *dReset* is toggled, *fHeight* should be reset to 0.

As mentioned in Chapter 3, the combinational part may only perform discrete state changes, on a topologically closed subset of $\mathcal{I} \times n \cdot \mathcal{S}$. This condition is satisfied by a HySChart defining the combinational part, if the precondition of every action in the chart identifies a topologically closed subset of $\mathcal{I} \times \mathcal{S}$. Note that in conjunction with hierarchy the action guards must be chosen with care in order to guarantee that the combinational part specified by the HySChart is total.

Events. Latched variables allow us to model many different communication styles. Particularly interesting for our example are *events* which we model by toggling boolean variables. The occurrence of an event is detected by testing if the current input value for that variable is different from the latched value of that variable, i.e., $e^{\backprime} \neq e$, where $e \in \mathbb{B}$ signals the occurrence of the event e. We write $e?$ as abbreviation for $e^{\backprime} \neq e \wedge e' = e^{\backprime}$. (The second part of the conjunction updates the latched value of e.) Similarly, sending an event is given by the following expression $e' = \neg e$ which is abbreviated by $e!$. With this notation, the filter reset action can be rewritten as $\mathit{dReset}? \wedge \mathit{fHeight}' = 0$. *Message passing* can be modeled equally easily [GSB98].

Preemption. In HySCharts we use transitions originating 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 [GSB98], one can define such a preemptive action to have higher priority than any action inside the hierarchic node (strong preemption) or to have lower priority than any action inside the node (weak preemption). Here, we use weak preemption, because it is simpler and better suited for the refinement of nodes. It allows that actions inside a hierarchic node overwrite the preemptive action.

The corresponding graph for a node with preemption is obtained as follows. Replacing a hierarchic node with preemptive actions $\mathtt{pa}_1, \ldots, \mathtt{pa}_h$ (Fig. 5.2, top left) by its corresponding graph of Figure 5.1, bottom right, yields a diagram of the form given in Figure 5.2, top right.[2] To obtain the semantics of the original node with the preemptive transitions, this diagram is in turn replaced by the graph in Figure 5.2, bottom.

This graph basically expresses that whenever a subnode is left on a wait exit point `wt` and one of the preemption actions is enabled, it is executed and followed by the exit action `exit` of the enclosing hierarchic node. The hierarchic node is then left along an exit point `pex` corresponding to the executed preemptive action. If none of the preemptive actions is enabled (`wait` is the negation of the disjunction of the guards of the $\mathtt{pa}_i$ actions) the hierarchic node is left along the wait exit point that corresponds to `wt` of the subnode.

In our example of Figure 1.2, left, the action *n2b* is a preemption action of the composed computation unit *noBend.*

The additive interpretation of graphs also provides the infrastructure to easily model history variables and other concepts known from statecharts-like formalisms [GSB98].

[2]The interior of the node is omitted here for clarity. It exactly is the graph of Figure 5.1, bottom right.

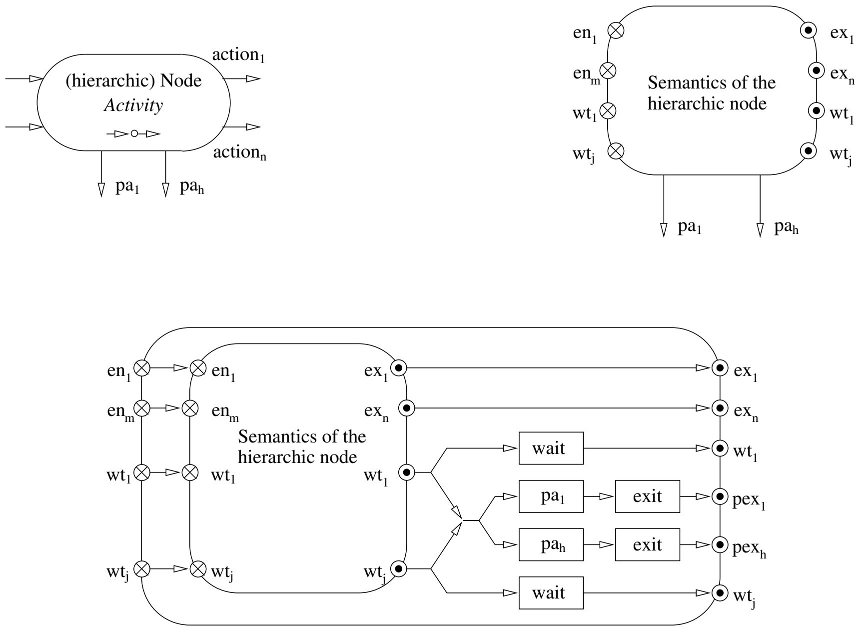

Figure 5.2: The semantics of preemption.

Semantics. If each node in the HySChart is replaced by the corresponding graph of Figure 5.1, right, and 5.2, right, we obtain a hierarchic graph whose nodes merely are relations. Writing the graph as the corresponding relational expression with the additive operators gives the denotational semantics of the HySChart's discrete part, i.e., the combinational 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. On the semantic level there is exactly one summand in the n-fold sum $n \cdot \mathcal{S}$ of the combinational part's type $(\mathcal{I} \times n \cdot \mathcal{S}) \to \mathcal{P}(n \cdot \mathcal{S})$ for every entry/exit point pair. The analog part uses the entry/exit point information encoded in this disjoint sum to select the right activity for every node in the HySChart (Section 5.2).

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

Example 4 (The EHC's Control component) We describe the states and transitions in Figure 1.2 in a top-down manner. The activities, written in italics in the figure, are explained in the next section.

The computation unit Control. On the top level of the component *Control* we have two computation units, *outBend* and *inBend*. When the controller senses that the car is in a curve, the computation unit *inBend* is entered. It is left again when the controller senses that the car no longer is in a curve. Sensing a curve is event-driven. We use the boolean variable *bend* for this purpose. The actions $n2b$ and $b2n$ are identical and very simple: $n2b \equiv b2n \equiv bend?$

The computation unit ***outBend*****.** The computation unit *outBend* is refined to *inTol* and *outTol* as shown in Figure 1.2, 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 then. If *fHeight* is outside this interval at a sampling point, one of the sub-nodes of *outTol* is entered. These sub-nodes are left again, when *fHeight* is inside the desired tolerance again and the filter is reset. The actions originating from *inTol* are defined as follows:

$$\begin{array}{ll} t\_o \equiv w = t_s, & i2i \equiv lb \leq fHeight \leq ub \\ i2u \equiv fHeight \leq lb, & i2d \equiv fHeight \geq ub \end{array}$$

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. Of course, one could have used three separate transitions instead. However, in this case the visual representation would have failed to highlight the common enabling condition t_o.

Leaving the computation unit *outTol* along its exit point *reset* causes the execution of the *reset* action. This action is always enabled and defined by $reset \equiv reset!$. Note that we used here the same name for the action and its associated event.

The transition $n2b$ originates from the composed node *outBend* (and from none of its sub-states). This expresses weak preemption, i.e., this transition can be taken from any sub-node of *outBend*, as long as it is not overwritten.

The computation unit ***outTol*****.** As shown in Figure 1.2, bottom right, the computation unit *outTol* consists of the computation units *up* and *down*. When the filtered chassis level is too low at a sampling point, node *up* is entered, where the compressor is on. When the level is too high, *down* is entered, where the escape valve is open. Control remains in these nodes until *fHeight* is inside the desired tolerance again (actions $u2i, d2i$). These actions cause *outTol* to be left along the same exit point, *reset*. The actions originating from *up* and *down* are very similar to those of *inTol*:

$$\begin{array}{lll} u2u \equiv fHeight \leq lb, & u2i \equiv lb \leq fHeight \leq ub, & u2d \equiv fHeight \geq ub, \\ d2d \equiv fHeight \geq ub, & d2i \equiv lb \leq fHeight \leq ub, & d2u \equiv fHeight \leq lb \end{array}$$

Again, ramification is used in the chart to highlight the common enabling condition t_o for these actions.

As indicated by the symbol $\multimap$ the nodes *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 nodes, i.e. all transitions directly originating from these nodes can only be taken at the end of a sampling interval.

Semantics. The combinational part follows directly from the HySChart by replacing the nodes by their corresponding graphs of Figure 5.1, right, and 5.2, right. As every wait entry/exit point pair at the highest level of the resulting graph corresponds to a summand in the type of the combinational part, we get that the combinational part of $Control$ has type:

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

Note that the user only has to draw the HySChart and give the definitions of the actions. The corresponding combinational part can be constructed automatically.

5.2 The Analog Part

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

Activities. Each activity name in the HySChart refers to a system of ordinary differential (in)equations over the variables of the component.[3] We demand that for any tuple of initial values $s \in \mathcal{S}$ and any continuous, infinitely smooth input stream $i \in \mathcal{I}^{\mathbb{R}_{c+}}$, the resulting initial value problem is solvable.

Example 5 (The activities of Control) In our example from Figure 1.2 the activity names written in italics stand for the following differential (in)equations:

$$\begin{array}{lcl@{\qquad}lcl} w\_inc & \equiv & \dot{w} = 1 & a\_inc & \equiv & \dot{c} \in [cp_-, cp_+] \\ a\_const & \equiv & \dot{c} = 0 & a\_dec & \equiv & \dot{c} \in [ev_-, ev_+] \end{array}$$

where $cp_-, cp_+ > 0$ and $ev_-, ev_+ < 0$ are constants. For w this means that it evolves in pace with physical time. Variable c either increases with a rate in $[cp_-, cp_+]$ (activity a_inc), it decreases (a_dec) or remains constant (a_const)

Note that this is all the user has to provide to specify the analog part. $\square$

The activity $Act \in (\mathcal{I} \times \mathcal{S})^{\mathbb{R}_{c+}} \to \mathcal{P}(\mathcal{S}^{\mathbb{R}_{c+}})$ in every node is derived from the differential (in)equations in the following way: For the input stream i and the state stream s we take $s(0)$ as the initial value for the system of differential (in)equations. The activity's set of output streams then consists of the solutions of the resulting initial value problem for input stream i. For those controlled variables v, whose evolution is not determined by the initial value problem, the activity's output is equal to $s.v$, i.e., to the v component of the state stream the activity received. Hence, it remains unmodified.

[3] An adaption of HySCharts to a more application specific syntax for activities, e.g. suited for multimedia streams, is feasible.

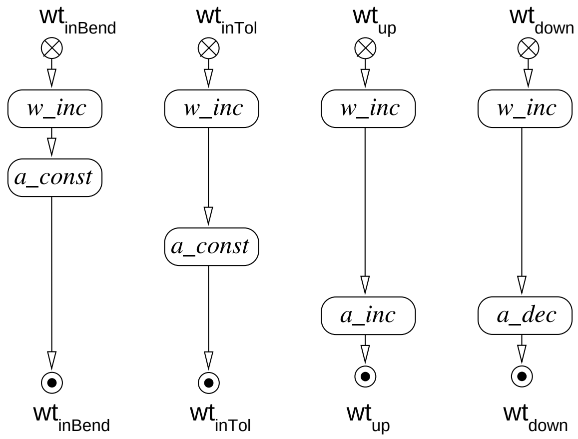

Figure 5.3: The $Control$ component's analog part.

Composition of Activities. To reflect the hierarchy in the HySChart the activities specified in the nodes are composed appropriately. Therefore, we extend the sequential composition operator ; to (disjoint sums of) activities:

$$Act_1 \,;\, Act_2 \quad = \{(i, \sigma, \sigma') \mid \exists \tau.\ (i, \sigma, \tau) \in Act_1 \wedge (i, \tau, \sigma') \in Act_2\}$$

A HySChart can be seen as a tree with the primitive nodes as its leaves. The HySCharts in Figure 1.2, 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, $CAct_N \stackrel{\text{def}}{=} Act_N$
- if N has sub-nodes $M_1, \ldots, M_n$ which have 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 primitive activities, then

$$CAct_N \stackrel{\text{def}}{=} +_{i=1}^{n}\ (+_{j=1}^{m_i}(Act_N; Act_{M_i,j}))$$

The analog part is the composed activity of the root node of the HySChart, i.e. $Ana = CAct_{root}$. Figure 5.3 and the following example explain this definition.

Example 6 (The analog part of Control) The HySChart in Figure 1.2 has the analog part:

$$\begin{aligned} Ana \quad \equiv \quad & (w\_inc\,;\, a\_const) + (w\_inc\,;\, \mathsf{I}\,;\, a\_const) + \\ & (w\_inc\,;\, \mathsf{I}\,;\, \mathsf{I}\,;\, a\_inc) + (w\_inc\,;\, \mathsf{I}\,;\, \mathsf{I}\,;\, a\_dec) \end{aligned}$$

where we applied associativity of ; and +. The activity names are used to refer to the semantics of each activity, here. Note that the expression is equivalent to

$(w\_inc\,;\,a\_const) + (w\_inc\,;\,a\_const) + (w\_inc\,;\,a\_inc) + (w\_inc\,;\,a\_dec)$, because the identity connector is the neutral element for sequential composition. Figure 5.3 depicts the analog part as a graph. □

The entry and exit point symbols in the figure highlight that the analog part has one path through the graph for every primitive node in the HySChart. When we construct the combinational 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 to sequentially compose the combinational part with the analog part as in the semantics of a hybrid machine in Chapter 3. The distinct wait points allow both the combinational part and the analog part to know which node in the HySChart currently has control and to behave accordingly.

In Chapter 3 we demanded that the analog part is *resolvable*. If the activities are defined as the solutions of solvable initial value problems as above, this is automatically ensured.

As a further example, Section 6.3 contains the HySChart for the EHC's filter component.

Chapter 6

Relation to Other Formalisms

6.1 Heterogeneous Component Specification

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 $\mathcal{I}^{\mathbb{R}_+} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}_+})$, where $\mathcal{I}$ and $\mathcal{O}$ is a set of input and output channels, respectively.

This means that any formalism can be used for component specification which defines a component as a total, time-guarded relation on piecewise smooth 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 7 (A delay element) The delay element of the EHC reproduces its input with delay $\delta > 0$. It can be specified directly as a relation as follows:

$$\begin{array}{lcl} D_f & \in & \mathbb{B}^{\mathbb{R}_+} \to \mathcal{P}(\mathbb{B}^{\mathbb{R}_+}) \\ D_f(i)(t) & = & \{false\} \text{ if } t < \delta \text{ and } \{i(t-\delta)\} \text{ otherwise} \end{array} \qquad \Box$$

6.2 Timed Automata

An interesting side effect of HySCharts is that they can simulate the non-urgent transitions of timed automata [AD94], although transitions in HySCharts are taken as soon as they are enabled. To explain this, Figure 6.1, left, shows a state of a timed automaton that must be left during the time interval $t \in [a, b]$. The equivalent node of a HySChart is given in Figure 6.1, right. The actions are $r \equiv x' = 0$ and $g \equiv x = 1$ and the activity is $Act \equiv \dot{x} \in [\frac{1}{b}, \frac{1}{a}]$. This means that instead of non-deterministically choosing a time instant between a and b, we non-deterministically choose a time scale between $[\frac{1}{b}, \frac{1}{a}]$. When the skewed clock x equals the limit 1 the transition is taken. E.g. for activity $\dot{x} = \frac{1}{b}$ we get that the transition is taken at $t = b$, because $x(t) = \frac{1}{b}\,t$.

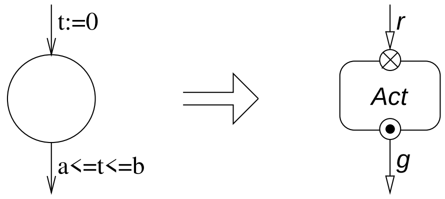

Figure 6.1: Translating timed automata into HySCharts.

Reading the translation from Figure 6.1 the other way round we get that a certain subclass of HySCharts, namely the one which only allows guards and activities of a form like those in the figure, can be translated into timed automata. Therefore, this class has a decidable reachability problem [ACH+95]. This result is comparable to the decidability of the *simple multirate timed systems* defined in [ACH+95].

6.3 A Typical Hybrid Component

Figure 6.2 shows the HySChart for the filter of the EHC example. Action name *set* stands for $dReset? \;\wedge\; fHeight' = 0$[1] and activity name f_follow denotes $\frac{d}{dt}fHeight = \frac{1}{T}(sHeight - fHeight)$, where T is the filter's time constant, i.e. a measure for its inertia.

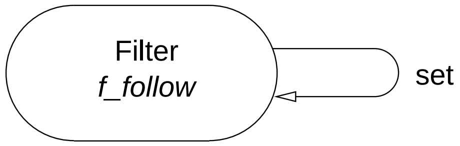

Figure 6.2: HySChart for the filter.

While both the discrete and the analog part of the filter are very easy, the component is nevertheless interesting from a hybrid point of view: 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 part that cannot exhibit discontinuities. Therefore, the filter underlines the need for hybrid specification techniques.

[1]Remember that $dReset?$ is a shorthand for $dReset^{\backprime} \neq dReset \;\wedge\; dReset' = dReset^{\backprime}$.

Chapter 7

Conclusion

Based on a clear hybrid computation model, we were able to show that the ideas presented in [GSB98] 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.

With an example we demonstrated the use of HyCharts and their features. Apart from many features known from statecharts-like formalisms, this in particular includes 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 to integrate description techniques from different engineering disciplines.

Methodically we conceive a HySChart as a very abstract and precise mathematical model of a hybrid system. Knowing exactly the behavior of the analog part as given by a system of differential (in)equations allows us to develop more concrete models that can easily be implemented on discrete computers. For such models it is essential to choose a discretization which preserves the main properties of the abstract description.

Although this paper mainly aims at hybrid systems appearing in the context of disciplines like electrical and mechanical engineering, we think that the continuous activities in HySCharts also make them well suited for specifying multimedia systems, such as video on demand systems. Basically HyCharts seem to be appropriate for any mixed analog/digital system where the use of continuous time is more natural than a discrete time model.

In the future we intend to develop tool support and a requirement specification language for HyCharts. For the verification of HySCharts we believe that the techniques known for linear hybrid automata [ACH$^+$95] can easily be adapted.

Acknowledgment. We thank Ingolf Krüger, Olaf Müller and Jan Philipps for their constructive criticism after reading draft versions of this paper.

Bibliography

[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.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AH97] R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *CONCUR 97: Concurrency Theory*, LNCS 1243. Springer-Verlag, 1997.

[Bro97] M. Broy. Refinement of time. In *ARTS'97*, LNCS 1231. Springer-Verlag, 1997.

[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.

[Eng89] R. Engelking. *General Topology*, volume 6 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1989.

[GSB98] R. Grosu, Gh. Stefănescu, and M. Broy. Visual formalisms revisited. In *Proc. Int. Conf. on Application of Concurrency to System Design (CSD)*. IEEE, 1998.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8, 1987.

[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*, volume 571 of *LNCS*, pages 591–619, Nijmegen, The Netherlands, 1992. Springer-Verlag.

[LSVW96] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[MS97] O. Müller and P. Scholz. Functional specification of real-time and hybrid systems. In *Proc. Hybrid and Real-Time Systems (HART)*, LNCS 1201. Springer, 1997.

[SGW94] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley and Sons Ltd, Chichester, 1994.

[SMF97] T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In *Proc. Hybrid and Real-Time Systems (HART'97)*, LNCS 1201. Springer-Verlag, 1997.

[Son90] E. D. Sontag. *Mathematical Control Theory*. Springer Verlag, 1990.

[Ste94] Gh. Stefănescu. Algebra of flownomials. Technical Report TUM-I9437, Technische Universität München, 1994.

[TMI98] The MathWorks Inc. Stateflow. `http://www.mathworks.com/products/stateflow/`, 1998.