T U M

I N S T I T U T F Ü R I N F O R M A T I K

Hybrid System Model

Thomas Stauner, Bernhard Rumpe, Peter Scholz

TUM-I9903
Januar 99

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-01-I9903-80/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Hybrid System Model[1]

Thomas Stauner, Bernhard Rumpe, Peter Scholz

Institut für Informatik, Technische Universität München

80333 München, Germany

http://www4.informatik.tu-muenchen.de/

Revised Version

[1]This paper emerges from a cooperation of the SysLab project, which is supported by the DFG under the Leibnizprogramm and by Siemens-Nixdorf, the BeQuest project, supported with funds of the DFG under reference number Br 887/9-1 within the priority program *Design and design methodology of embedded systems*, and the FORSOFT project.

Abstract

In this paper we provide a mathematical characterization of systems, that contain analog components, as well as strongly time depended control components and rather time independent information processing components. The presented system model is based on the notion of streams and stream processing functions. It is used to formalize and integrate the semantics of different description techniques that occur in disciplines like computer science, economics, electrical and electronical engineering, and mechanical engineering.

In the paper we develop a hierarchy of models. We start with a core model that allows us to describe both discrete and continuous, that is, hybrid behavior. Abstractions and refinements of this core model allow to connect to given models for discrete timed and untimed systems. The system model presented here is intended to be the base layer for other system models.

Contents

Symbol Glossary

- $\mathbb{N}$: the natural numbers
- $\mathbb{R}_+$: the non-negative real numbers
- $x(t)$: the value transmitted on dense stream or stream tuple x at time $t \in \mathbb{R}_+$
- $x{\downarrow}t$: the restriction $x|_{[0,t]}$ for the dense stream or stream tuple x
- $x^{(n)}$: the n-th time derivative of function x
- $\mathbb{C}$: the set of channel identifiers
- $\mathbb{M}$: the data universe
- $\mathbb{M}_c$: channel type for channel $c \in \mathbb{C}$ with $\mathbb{M}_c \subseteq \mathbb{M}$ and $\mathbb{M}_c \neq \emptyset$
- $\varepsilon \in \mathbb{M}$: special value to denote the absence of a message
- $\mathbb{SM}$: the mathematical model or system model
- $\mathbb{SM}_1$, $\mathbb{SM}_2$: previous versions of the system model
- $\mathbb{SM}_3$: the new version of the system model, discussed in this contribution
- $\overrightarrow{c} \in \mathbb{M}_c^{\mathbb{R}+}$ the set of dense streams of channel c that obeys the channel typing
- $\overset{\ominus\!\!\rightarrow}{c} \subseteq \overrightarrow{c}$ the kind of channel c
- $\overrightarrow{C} = \{x : C \to \mathbb{M}^{\mathbb{R}+} \mid \forall c \in C : x(c) \in \mathbb{M}_c^{\mathbb{R}+}\}$, the set of named stream tuples with domain C that in addition obey the channel typing
- $\overset{\ominus\!\!\rightarrow}{C} = \{x \in \overrightarrow{C} \mid \forall c \in C : x(c) \in \overset{\ominus\!\!\rightarrow}{c}\}$ the set of named stream tuples with channel kinds and domain C
- $\overrightarrow{I} \xrightarrow{0} \overrightarrow{O}$: the set of deterministic behaviors without delay

- $\overrightarrow{I} \overset{\delta}{\rightarrow} \overrightarrow{O}$: the set of delayed deterministic behaviors with delay δ

- $\overrightarrow{I} \overset{dly}{\rightarrow} \overrightarrow{O}$: the set of delayed deterministic behaviors each with delay, but no lower bound

- $\overset{\ominus\!\!\!\rightarrow}{I} \overset{\delta}{\rightarrow} \overset{\ominus\!\!\!\rightarrow}{O}$: the set of delayed deterministic behaviors with delay δ that respects the kind of its channels

- $\wp(\overrightarrow{I} \overset{0}{\rightarrow} \overrightarrow{O})$: behaviors without delay

- $\wp(\overrightarrow{I} \overset{\delta}{\rightarrow} \overrightarrow{O})$: delayed behaviors with delay δ

- $\wp(\overset{\ominus\!\!\!\rightarrow}{I} \overset{\delta}{\rightarrow} \overset{\ominus\!\!\!\rightarrow}{O})$: the set of delayed behaviors with delay δ that respects the kind of its channels

- δ: positive real constant

- $\delta(f) = sup\{\varrho \in \mathbb{R}_+ \mid \forall x, y \in \overrightarrow{I}, t \in \mathbb{R}_+ :$
  $x \downarrow t = y \downarrow t \Rightarrow f(x) \downarrow (t + \varrho) = f(y) \downarrow (t + \varrho)\}$, the maximal delay of function f

- $\delta(F) = inf\{\delta(f) \mid f \in F\}$, the maximal delay that holds for behavior F

- $\otimes$: the nested composition operator $\otimes_{k \in K} F_k$

- $F \updownarrow_J^P$: the behavior derived from behavior F with adapted interface $\wp(\overrightarrow{J} \overset{\delta}{\rightarrow} \overrightarrow{P})$, that is, $g \in F \updownarrow_J^P \iff \exists f \in F : \forall x \in \overrightarrow{J} : g(x) = f(x|_I)|_P$. Input I is extended to J; the output is restricted to P

- $;B_K = (\otimes B_K) \updownarrow_{I_0}^{O_n}$, the strict sequential composition

- $\uparrow F = \otimes\{F\} \updownarrow_I^O$, the feedback of behavior F on channels common to input and output of F

- $D(x)$: with $D(x) \subseteq \mathbb{R}_+$, the set of all points in time, where stream x is not smooth, i.e. it exhibits a discontinuity

- M^*: the set of all finite sequences of messages in M

- M^∞: the set of all infinite sequences of messages in M

- $M^\omega = M^\infty \cup M^*$

- $. \frown . : M^\omega \times M^\omega \rightarrow M^\omega$: the concatenation of two streams, i.e. the stream which is obtained by putting the second argument after the first; $\frown$ will also be used to concatenate a single message with a stream

- $\# : M^\omega \rightarrow \mathbb{N} \cup \{\infty\}$: gives the length of a stream as a natural number or ∞, if the stream is infinite
- $©: \wp(M) \times M^\omega \rightarrow M^\omega$: the filter-function; $©(N, s)$ deletes all elements in s which are not contained in set N
- ∇_T: the operator to transform dense streams to (discrete) timed streams
- ∇: the operator to transform discrete timed streams to untimed streams
- $\overline{c} = \mathbb{M}_c^\omega$ the set of untimed streams of channel c (abstraction, only for certain kinds of channel types)
- $\tilde{c} = (\mathbb{M}_c^*)^\infty$ the set of timed streams of channel c (abstraction, only for certain kinds of channel types)
- $\tilde{I} \overset{n}{\rightarrow} \tilde{O}$: the set of timed (discrete) stream processing functions with delay n (natural number)
- $\overline{I} \overset{utd}{\rightarrow} \overline{O}$: the set of untimed (discrete) stream processing functions
- $(\overrightarrow{C}, d_C)$ the metric space of named stream tuples
- $(\overset{\ominus\!\!\!\rightarrow}{C}, d_C)$ the metric space of named stream tuples with channel kinds

1. What a System Model is about

What we call a *system model* is a model for software systems *and* their environment. Depending on the kind of system, the environment may be a hardware device, a physical plant in the case of embedded systems, or an enterprise in the case of business information systems. Usually a system model does not cover all aspects of the modeled systems. Instead it is an abstract view representing those aspects of the modeled systems that are considered to be relevant for their functionality. The mathematical formulation of this view provides an unambiguous semantics that leads to a better understanding of the entire system under development.

1.1. Description Techniques and their Semantics

When describing software systems as well as hardware systems or combinations like embedded systems a notation is necessary. Today lots of different description techniques, including the ones defined by UML [Gro97b], Petri Nets [Rei90], StateCharts [Har87], SDL [Hog89], MSC [IT96], and for continuous systems differential equations and the like, have been developed. Although these notations usually do have an intuitive meaning, the intuition of the users of such notations often differs significantly in some points from the semantics of these description techniques.

One of the best possibilities to improve this situation and to help to better understand the description techniques in detail is to define a formal semantics for a notation. However, we should not forget that such a formal semantics in itself is by no means suited for comprehending and understanding a notation [Rum98]. In order to learn the usage of a notation it is necessary to grasp its intuitive meaning. This can be achieved better by applying it to toy problems than by reading mathematical formulae. Hence, the formal semantics is just a means to aid different users of a notation in having the same intuitive understanding. Moreover, a formal semantics guarantees that this intuitive understanding coincides with the operational behavior of those tools that implement the notation.

Tools could greatly benefit from a precise semantics definition, because a semantics allows to make precise statements on the way in which a syntactic transformation in a specification affects the behavior of the specified system. Thus, tools could provide a set of transformations, which guarantee that after each application of a transformation, certain aspects of the system's behavior will be unchanged. This would enable the user not only to manipulate diagrams, but also to manipulate them in a way that ensures that certain properties remain valid, without having to prove this explicitly on the semantic level with complicated formulae. A formal semantics is the prerequisite for applying, possibly automated, techniques that ensure or check the correctness of a system.

So far, we have not explained what we understand by the notion of *semantics*. The problem of giving a semantics to a syntactic language $\mathbb{L}$ has been tackled in compiler theory, semantics of programming languages and especially the theory of algebraic specifications [Wir90, BFG+93] for many years. Recently these approaches have also been applied to diagrammatic description techniques like the UML [BHH+97, FELR98, FELR97].

According to the results of compiler theory, the definition of a semantics in a denotational style for a language is given by:

1. defining the syntax $\mathbb{L}$ of the language,
2. choosing the semantics domain, i.e. the mathematical model (or *system model*) which we will denote by $\mathbb{SM}$, and
3. defining a semantics mapping $[\![.]\!] : \mathbb{L} \to \mathbb{SM}$.

For traditional programming languages, the language $\mathbb{L}$ is purely textual, and can therefore be described by Chomsky grammars. These grammars can be used to describe the concrete syntax as well as the abstract syntax (without syntactic sugar) and, furthermore, to construct parsers.

Today, many of the description techniques used in practice apply a visual notation. They exhibit a two dimensional structure whose elements are attributed with textual parts. At present, it is still a challenge to find an appropriate technique to describe the syntax of such notations. Two important approaches to this problem are graph grammars and meta-models, as e.g. used in UML [Gro97a]. Graph grammars are an extension of ordinary grammars, capturing the two dimensional structure of a visual formalism. Their development is still under progress, and unfortunately they are still not as conveniently usable as their linear counterparts. A meta model is basically a Class Diagram that describes the abstract syntax of a notation. In UML meta models are used for two reasons. First, the users of UML are familiar with Class Diagrams (as it is a notation within UML) and therefore need not learn another notation. Second, the meta model is the principle implementation technique for tool vendors (and UML developers are to a large extent tool vendors). However, such meta models have severe short-comings. For instance, it is hardly possible to define context conditions like “all relations are acyclic”. Furthermore, meta models are not very abstract, and it is complicated to define a mathematical semantics based on them.

In this paper, we will neither assume a specific way to describe the syntax of a language or description technique $\mathbb{L}$ nor will we further discuss graph grammars and meta models. Instead, we will concentrate on elaborating a mathematical model that is general enough to serve as a semantical basis for a broad variety of description techniques.

A system model is therefore not intended to be given directly to the developer, but only serves as implicit semantics for the notation the developer actually manipulates.

1.2. How to Describe and Use a System Model

A system model describes the properties of the kind of systems in which we are interested. For example, it precisely captures basic assumptions about:

- the communication style between the system components (asynchronous or synchronous),
- the underlying object model and kind of inheritance (if any),
- the time model (continuous time, discrete time, or untimed),
- the system structure (fixed or dynamic).

Often, such assumptions are only made implicitly when a semantics is defined. Therefore, it is rather difficult to figure out differences between various definitions of a semantics. Hence, we decided to make such assumptions *as explicit as possible*. Apart from that, our aim was to find a proper set of basic assumptions that is on the one hand rather free, allowing e.g. different styles of communication, but on the other hand also gives us a set of refinement and composition techniques at hand. These basic assumptions are given in Section 2, in which the basic system model is defined.

Please note once more that the system model is not intended to serve as toolbox for software engineers to reason about their designs. Instead, it is intended to allow method developers and tool vendors to reason about their methodical guidelines and tool manipulations and to improve those guidelines and manipulations with respects to the result of this reasoning.

The software engineer, who applies tools and methods based on the model, does not need to understand the system model precisely. Based on his intuitive understanding of the kind of systems he develops, an informal, motivating description suffices. However, this description has to be precise enough to grasp the underlying ideas of the system model.

The basic properties of all systems we are interested in are captured in the following chapter by defining axioms. Of course, the formalization of these properties affects the development of systems. There is a trade-off between the idea of defining everything as detailed as possible, with the effect that the system model does not allow to define the semantics of a certain description technique, or leaving certain details unspecified. Furthermore, the more sophisticated description techniques are, the more complex their semantics usually is. A system model not only has to capture input/output behavior of a system, but also structural and datatype issues.

Formally, the *system model* $\mathbb{SM}$ is the set of systems that obeys the properties defined in Chapter 2. As this version of the system model originates from two prior models and some further work as outlined in Section 1.3, we call it $\mathbb{SM}_3$:

$$\mathbb{SM}_3 = \{sys \mid \mathbb{SM}_3(sys)\}.$$

We do not define the characteristics for properties of $\mathbb{SM}_3$ as one explicit predicate, but formulate them as a set of axioms.

In principle, a *system sys* can be formalized as a tuple, whose components capture information such as the identifiers used in the system, ID_{sys}, or the typing of the variables in the system. A concrete specification of a system or a set of systems can be interpreted as an assignment of concrete sets, functions and relations to the components of the tuple (comparable to the definition of Σ-algebras in accordance with a signature σ).

The axioms given in the next chapter formally speak about a system sys or components of its characterizing tuple, respectively. For notational convenience, we usually omit suffix sys when referring to the tuple's components.

1.3. History of this System Model

It is a major aim of several research projects within our institute to define a coherent set of description techniques, like Class Diagrams, State Transition Diagrams and others, and to give them an integrated formal semantics. It is the integration of the semantics that allows us to formulate precise context conditions for heterogeneous documents. Moreover, a system model is the basis for provably correct translations of one document into another, for instance the generation of code from state diagrams.

The system model presented in this paper results from a synthesis of the work on object-oriented system models within the SysLab project, the work on embedded real-time systems within the FORSOFT project and the work on hybrid (and real-time) systems within the BeQuest project. The model is based on two earlier versions developed within SysLab.

The first version for this system model was defined in [RKB95] and contained a strong motivational part. This system model was further improved in [KRB96]. From now on, we refer to this first version of the system model as $\mathbb{SM}_1$.

The second version, $\mathbb{SM}_2$, was defined in [GKR96]. It clarifies some details and enhances $\mathbb{SM}_1$ by port automata, which were introduced in [GR95]. While [GKR96] contains much technical details, it is nevertheless illustrated with two examples: a dynamic *queue* and a *RS-flip-flop*. Thus, it shows that the system model is capable of formalizing highly dynamic processes as well as digital hardware systems, and combinations thereof. $\mathbb{SM}_2$ also relaxes the communication between components to a more general concept, allowing different mechanisms ranging from *broadcasting* to the point-to-point communication used hitherto. Both $\mathbb{SM}_1$ and $\mathbb{SM}_2$ where combined and enhanced in [Rum96], where the idea of using an integrated system model to define the semantics of several description techniques also emerged.

In this paper, we define $\mathbb{SM}_3$ which is based on continuous and discrete communication and we relate this new model to the already existing ones by providing appropriate abstraction functions. We also add an individual typing of channels and introduce different types or kinds of channels. These kinds include the timed message channels used so far, but also allow signal channels, which contain sets of messages at a time, and channels with continuously varying values as used in control systems. In contrast to $\mathbb{SM}_2$, we do not include hierarchy and internal states in the basic layer of $\mathbb{SM}_3$. Furthermore, $\mathbb{SM}_2$ can be seen as an abstraction from $\mathbb{SM}_3$ to discrete streams.

2. Specifying with Stream Processing Functions

2.1. Motivation

As outlined in the introduction, the primary aim of this paper is to develop a structured system model that is suitable both for integrating different description techniques from a single application area as well as for integrating description techniques from different disciplines. The disciplines we have in mind are computer science, economics, electrical and electronical engineering, and mechanical engineering. In some of these areas time is dense in others people think in discrete time steps and yet in others time is not considered at all. Likewise, in all these areas system components exchange different types of data.

Thus, in order to create a hierarchy of domain specific models, we chose a core model with a dense time scale, i.e. time is a non-negative real number, and allow arbitrary data domains. Dense time is used, because it is a very general model of time. This way the core model can easily be adopted to more specific settings.

As noted already in the previous section, the following set of axioms characterizes a set of systems, denoted as $\mathbb{SM}_3$, where each system $sys \in \mathbb{SM}_3$ consists of a tuple of elements, like the set of channel identifiers $\mathbb{C}_{sys}$. For reasons of clarity, we omit the suffix in the following.

2.2. Preliminaries

We regard a distributed system as a network of components communicating via directed channels. Communication occurs without delay, but the components may exhibit delay. On every input or output channel data, e.g. messages or signals, are received from or sent to the environment. Therefore, every channel reflects an input or output communication history of the system. A deterministic system can be described by one function for each component. Each function processes input histories and produces output histories according to its specification. To describe under-specification or nondeterminism we can specify each component by a set of functions, called a *behavior*, instead of a single function.

2.3. Dense Communication Histories

A communication history can be modeled as a *dense stream*, i.e. as a total function $x : \mathbb{R}_+ \to M$, where $\mathbb{R}_+$ denotes the set of all non-negative real numbers and M is the (potentially infinite) non-empty data domain. At each moment of time t, $x(t)$ denotes the value transmitted on channel x at this time.

In order to enable one to describe real-time and hybrid reactive systems, which continuously respond to stimuli from the environment, we use a dense time scale. As reactive systems are not designed to terminate when any result is achieved, but to stay continuously active after being started, we use whole $\mathbb{R}_+$, not just a finite interval, as the time scale. In Chapter 4, several application specific specializations of this general stream model will be introduced. In particular, we will encounter the well-known discrete streams [Bro93, BDD+93], which use the natural numbers $\mathbb{N}$ instead of $\mathbb{R}_+$ as their underlying time scale.

In the following, we write $M^{\mathbb{R}_+}$ for the set of all dense streams over set M. For every dense stream x we abbreviate the restriction $x|_{[0,t]}$ by $x \downarrow t$. Appendix A shows that $M^{\mathbb{R}_+}$ is a complete metric space with respect to the metric defined there. This result is needed to be able to deal with feedback loops in functional specifications.

2.4. Named Stream Tuples

Usually, components not only have one input and one output channel, but several ones. Therefore, we assume that there is a given set of channel identifiers $\mathbb{C}$, and a given data universe $\mathbb{M}$. As we use different kinds of channels, we introduce types for channels. Each channel $c \in \mathbb{C}$ has its own channel type $\mathbb{M}_c \subseteq \mathbb{M}$, $\mathbb{M}_c \neq \emptyset$.

Given a set of channel names $C \subseteq \mathbb{C}$ a *named stream tuple* is a function $C \to (\mathbb{R}_+ \to \mathbb{M})$ that assigns communication histories to channel names. We write $\overrightarrow{C}$ for the set of named stream tuples with domain C that in addition obey the channel typing. Therefore:

$$\overrightarrow{C} = \{x : C \to \mathbb{M}^{\mathbb{R}_+} \mid \forall c \in C : x(c) \in \mathbb{M}_c^{\mathbb{R}_+}\}$$

For $x \in \overrightarrow{C}$ and $C' \subseteq C$, the named stream tuple $x|_{C'} \in \overrightarrow{C'}$ denotes the restriction of x to the channels in C':

$$\forall c \in C' : x|_{C'}(c) = x(c)$$

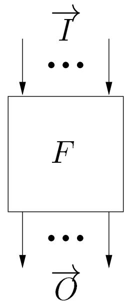

Figure 2.1.: Stream Processing Function.

Furthermore, we allow to add named stream tuples like functional adding by $(x + x')$, provided $x \in \overrightarrow{C}$ and $x' \in \overrightarrow{C'}$ coincide on $C \cap C'$, i.e. $x|_{C'} = x'|_C$. Moreover, $(x + x')(c) = x(c)$ if $c \in C$ and $(x + x')(c) = x'(c)$ if $c \in C'$.

Note that, due to currying, we regard the domains $C \to (\mathbb{R}_+ \to \mathbb{M})$ and $\mathbb{R}_+ \to (C \to \mathbb{M})$ to be isomorphic and sometimes will make use of that. For instance we may overload the cut-operator $x{\downarrow}t$ to stream tuples in a pointwise style, resulting in a mapping $C \to ([0, t] \to \mathbb{M})$, which is abbreviated as $\overrightarrow{C}[0, t]$. We speak of the *time domain* $[0, t]$ resp. $\mathbb{R}_+$ and of the *channel domain* C of such a function.

We overload channel restriction $|$ and addition $+$ of stream tuples to time restricted streams, but require when adding two time restricted tuples that both have the same time domain. Moreover, we extend our operators to sets of named stream tuples, by applying them in an elementwise style, e.g. for a set S of named stream tuples, $S{\downarrow}t = \bigcup_{s \in S} s{\downarrow}t$.

2.5. Stream Processing Functions

Components of real time or hybrid systems can be functionally specified by stream processing functions over dense streams [MS97]. First ideas in this area come from system theory [MT75]. Components are connected by directed channels to form a network. An (I, O)-stream processing function with input channels I and output channels O is a function f with the type:

$$f : \overrightarrow{I} \to \overrightarrow{O}$$

The structure of a component including its interface can be pictured as shown in Fig. 2.1.

Our operational understanding that stream processing functions model interacting components leads to a basic requirement for them: An interactive component is not capable of undoing an output that it has sent already. This requirement can be fulfilled by a certain kind of stream processing functions, which we call *deterministic behaviors.*

A stream processing function is said to be a *deterministic behavior* if its input until time t completely determines its output until time t. It is said to be a *delayed deterministic behavior* if its input until time t completely determines its output until time $t + \delta$ for an existing, time independent $\delta > 0$. In other words, a delayed deterministic behavior imposes a delay of at least an arbitrarily small real value between input and output. Here, δ denotes the delay of f.

It is quite realistic to assume components to be delayed because reactive systems always need a certain time to react. However, it is not always useful to explicitly deal with delay. E.g. there are system engineering techniques, that deal with instantaneous reactions, disregarding any delay, until it comes to an implementation. It is therefore useful to define and reason with non-delayed behaviors, and later-on add delay if necessary. Another viewpoint, which leads to the same methodical treatment is to disregard delays at first and just consider delay, when appropriate.

Definition 1 ((Delayed) Deterministic Behavior) *An* (I, O)*-stream processing function* $f : \overrightarrow{I} \to \overrightarrow{O}$ *is called a* deterministic behavior *if*

$$\forall x, y \in \overrightarrow{I}, t \in \mathbb{R}_+ : x{\downarrow}t = y{\downarrow}t \Rightarrow f(x){\downarrow}t = f(y){\downarrow}t$$

and a delayed deterministic behavior (with delay $\delta > 0$) *if*

$$\forall x, y \in \overrightarrow{I}, t \in \mathbb{R}_+ : x{\downarrow}t = y{\downarrow}t \Rightarrow f(x){\downarrow}(t+\delta) = f(y){\downarrow}(t+\delta).$$

We denote the set of deterministic behaviors with input channels I *and output channels* O *by* $\overrightarrow{I} \stackrel{0}{\to} \overrightarrow{O}$ *and the set of delayed deterministic behaviors with delay* δ *by* $\overrightarrow{I} \stackrel{\delta}{\to} \overrightarrow{O}$.

Furthermore, the set of all delayed deterministic behaviors (without any lower bound) is denoted as $\overrightarrow{I} \stackrel{dly}{\to} \overrightarrow{O}$, which can be characterized as $\bigcup_{d>0} \overrightarrow{I} \stackrel{\delta}{\to} \overrightarrow{O}$. Unfortunately this kind of behaviors is not closed under certain operators, like nested composition if infinitely many components are involved, and will therefore not be considered further.

For $\delta_1 \geq \delta_2$ it naturally holds that $(\overrightarrow{I} \stackrel{\delta_1}{\to} \overrightarrow{O}) \subseteq (\overrightarrow{I} \stackrel{\delta_2}{\to} \overrightarrow{O})$. Therefore, maximal delay $\delta(f)$ can be attached to each behavior function f. Formally:

$$\begin{aligned} \delta(f) = \quad & sup\{\varrho \in \mathbb{R}_+ \mid \forall x, y \in \overrightarrow{I}, t \in \mathbb{R}_+ : \\ & \qquad x{\downarrow}t = y{\downarrow}t \Rightarrow f(x){\downarrow}(t+\varrho) = f(y){\downarrow}(t+\varrho)\} \end{aligned}$$

A function with $\delta(f) = +\infty$ does not depend on its input at all and therefore behaves like a constant function, i.e. it yields the same output stream for every input stream.

2.6. Sets of Stream Processing Functions

A stream processing function describes a deterministic behavior: for each single input communication history exactly one output communication history is provided. In order to allow non-deterministic behavior of components, we do not deal with single functions, but with sets of stream processing functions. The component is then required to obey one of these functions. From an observational point of view, it makes no difference, whether this choice takes place initially or iteratively during runtime.

Definition 2 ((Delayed) Behavior) *A non-empty set of (I, O)-stream processing functions $F \in \wp(\overrightarrow{I} \to \overrightarrow{O})$ is called a* behavior *if each function $f \in F$ is a deterministic behavior. It is called* a delayed behavior (with delay $\delta > 0$) *if for each function $f \in F$, it holds that $\delta(f) \geq \delta$. We denote the set of behaviors by $\wp(\overrightarrow{I} \xrightarrow{0} \overrightarrow{O})$ and the set of behaviors with delay $\delta > 0$ by $\wp(\overrightarrow{I} \xrightarrow{\delta} \overrightarrow{O})$.*

Given a behavior F, we denote the maximal delay that holds for F by $\delta(F)$. Formally, $\delta(F)$ is the infimum of the supremal delays of the $f \in F$:

$$\delta(F) = inf\{\delta(f) \mid f \in F\}$$

A behavior with $\delta(F) = +\infty$ does not depend on its input at all, but still has the chance to behave differently in different system runs. For example a random number generator exhibits such a behavior.

Deterministic behaviors $f : \overrightarrow{I} \xrightarrow{\delta} \overrightarrow{O}$ can be considered as implementation strategies, whereas nondeterministic behaviors $F \in \wp(\overrightarrow{I} \xrightarrow{\delta} \overrightarrow{O})$ can be seen as specifications.

Using sets of functions is a rather flexible, but fine grained approach to describe behaviors of components. Based on the experience gained so far, it is not recommended to the developer to use sets of stream processing functions directly. Instead notations should be used that abstract away from several details, and also from some of this complexity. One first simplification would be the use of relations of the kind $F : \overrightarrow{I} \xrightarrow{\delta} \wp(\overrightarrow{O})$. However, even this abstraction hides some important details. Namely it is the case that even if the relation F exhibits some delay $\delta > 0$, its deterministic descendants $f \in F$ need not (see Example 1). Therefore, as a further constraint an explicitly given delay is needed when extracting functions from relations.

Example 1 (Relations and delay) Let $R : \overrightarrow{I} \xrightarrow{\delta} \wp(\overrightarrow{O})$ be the relation defined as $R(i) = \wp(\overrightarrow{O})$ for all $i \in \overrightarrow{I}$. We could assign it a delay of $\delta(R) = \infty$, as no reaction to the input is visible in this relation at all. As this is the full relation, any other relation is a subset of R. Therefore also the identity function id, emitting instantly its inputs again, is a deterministic descendant of R. The identity function, however, has a delay of $\delta(id) = 0$.

In the functional world R would correspond to the set of functions $F = \wp(\overrightarrow{I} \xrightarrow{0} \overrightarrow{O})$ with a delay $\delta(F) = 0$. $\square$

3. Composition Operators

The definition of networks is the main structuring principle for distributed systems. To allow the hierarchical composition of components, it is necessary to not distinguish between a single component and a network of components. A network can be defined in two equivalent styles, either by recursive equations or by special composition operators. We choose the second alternative and consider a very general nested composition operator $\otimes$, which was first introduced for the discrete time layer in [PR97]. This operator can be adapted to at least three common specializations, namely *sequential* and *parallel composition* and *feedback*.

Please note that $\otimes$ is a mathematical operator, i.e. it is defined in mathematical terms, not a specifications operator, as we do not provide a concrete syntactic notation here, which the operator could be part of. Nevertheless there is a specification methodology given in FOCUS that provides such operators [BDD+92].

3.1. Nested Composition

Nested composition $\otimes$ relies on the idea that a set of given components is connected to form a subsystem by relating interfaces with common names. Let B_K be a set of behaviors

$$B_K = \{F_k \in \wp(\overrightarrow{I_k} \xrightarrow{\delta_k} \overrightarrow{O_k}) \mid k \in K\},$$

where K is a possibly infinite index set and all components have disjoint output channels, formally $\forall k, j \in K : k \neq j \Rightarrow O_k \cap O_j = \emptyset$.

We compose these behaviors in parallel with implicit feedback. The signature of the composition $\otimes B_K \in \wp(\overrightarrow{I} \xrightarrow{\delta} \overrightarrow{O})$ is given as:

$$O = \bigcup_{k \in K} O_k, \qquad I = (\bigcup_{k \in K} I_k) \setminus O$$

The set O of output channels is the union of all output channels and I is the set of those input channels that are not connected to any of the components' outputs. Thus, I are the remaining inputs, which are still part of the interface. In particular $I \cap O = \emptyset$.

Then the behavior $\otimes B_K \in \wp(\overrightarrow{I} \stackrel{\delta}{\rightarrow} \overrightarrow{O})$ is characterized by the following mathematical definition:

$$\begin{aligned} f \in (\otimes B_K) \Leftrightarrow \quad & \forall k \in K : \exists f_k \in F_k . \\ & \quad \forall i \in \overrightarrow{I}, o \in \overrightarrow{O}. \\ & \qquad o = f(i) \Rightarrow o|_{O_k} = f_k((i+o)|_{I_k}) \end{aligned}$$

The composition is not well defined for all cases. If no delay is involved, instantaneous feedback may occur and lead to an undesirable result: As we characterized composition in a functional style, the resulting set of functions does not need to be a behavior at all. It may be empty or it may include behaviors that are more or less chaotic after some singularitan point has been reached.

A requirement that is clearly sufficient for the resulting behavior $\otimes B_K$ to be meaningful, is that all composed behaviors are delayed with the infimum of their delays $\mathit{inf}\{\delta(F_k)|k \in K\}$ being greater than 0 (see Appendix B.1). This is not really a restriction, as instantaneous behavior does not occur in reality, but is only an abstraction that allows to simplify some thoughts and arguments about such systems. Statecharts, for instance, do not speak about time delay, but use a concept of micro- and macro steps instead [vdB94]. Nevertheless, weaker conditions, i.e. less delayed behaviors, are possible. It is for example already sufficient to ensure that no feedback loop without delay occurs. In the subsequent sections three specializations of $\otimes$ are introduced and we will see that a delay is necessary for one of them only, namely for the feedback operator.

If all behaviors in B_K are delayed and the infimum of their delays $\mathit{inf}\{\delta(F_k)|k \in K\}$ is greater than 0, we can prove that the resulting behavior $\otimes B_K$ is delayed with $\delta(\otimes B_K) \geq \mathit{inf}\{\delta(F_k)|k \in K\}$ (see Appendix B.1).

Please note that nested composition is rather general. It allows one behavior to be composed (simple feedback), as well as an infinite number of behaviors to be composed. Furthermore, it allows channels to be shared among several readers.

For deterministic stream processing functions, we can provide the same operator using nearly the same definition. If delayed stream processing functions whose infimum of their delays is greater than zero are composed, the result is a deterministic stream processing function again. (See the proof in Appendix B.1.)

In fact, if we select a function f_k from each behavior F_k, then the composed function implements the composed behavior: $\otimes b_K \in \otimes B_K$, where $b_k = \{f_k|k \in K\}$. This immediately results from the fact, that the composition $\otimes$ is monotonic versus refinement: For $B_K = \{F_k|k \in K\}$ and $D_K = \{G_k|k \in K\}$, if for all $k \in K$ it holds that $F_k \subseteq G_k$ then $\otimes B_K \subseteq \otimes D_K$. (See the proof in Appendix B.1.)

3.2. Renaming

A renaming operator is necessary if we want to compose behaviors with sequential composition, parallel composition and feedback instead of using nested composition. This may be desirable if not all of the behaviors to be composed are delayed.

In order to define renaming, we need an injective *renaming function* $R \in \mathbb{C} \to \mathbb{C}$ on the channel identifiers $\mathbb{C}$ that obeys the channel typing:

$$\forall c \in \mathbb{C} : \mathbb{M}_c = \mathbb{M}_{R(c)}$$

R is then overloaded to several other constituents, like stream tuples, behaviors, etc. A renamed set of named stream tuples is denoted as $\overrightarrow{R(C)}$, where R is extended to sets of channel identifiers in a pointwise manner.

For a named stream tuple $x \in \overrightarrow{C}$ the renamed tuple $R(x) \in \overrightarrow{R(C)}$ is defined by $R(x)(R(c)) = x(c)$. The renaming functions on named stream tuples may not identify different channels in the stream tuple. This is guaranteed by the injectivity of the renaming function.

For a tuple $R = (R_I, R_O)$, consisting of a renaming function for the input channels and one for the output channels of behavior $F \in \wp(\overrightarrow{I} \overset{\delta}{\to} \overrightarrow{O})$, the renamed behavior $R(F) \in \wp(\overrightarrow{R_I(I)} \overset{\delta}{\to} \overrightarrow{R_O(O)})$ can be defined as:

$$R(F)(R_I(x)) = R_O(F(x))$$

3.3. Interface Adaption

Besides renaming there is another helpful operator that allows to adapt the interface of behaviors. Interface adaption allows to hide output channels that are not needed anymore and to add input channels that have not been used so far. Hiding is a commonly used technique to allow encapsulation. The opposite is done by interface extension, which allows to plug in new information and reduce nondeterminism that the behavior had before. Given a behavior $F \in \wp(\overrightarrow{I} \overset{\delta}{\to} \overrightarrow{O})$ and an extension $J \supseteq I$, as well as a restriction $P \subseteq O$, then by $F \updownarrow_J^P$ we denote the new behavior with adapted interface $\wp(\overrightarrow{J} \overset{\delta}{\to} \overrightarrow{P})$. It is defined by:

$$g \in F \updownarrow_J^P \quad \Leftrightarrow \quad \exists f \in F : \forall x \in \overrightarrow{J} : g(x) = f(x|_I)|_P$$

The definition is given in such a way that newly added input channels do not contribute to the component's current behavior, but can be used for later consideration.

3.4. Sequential Composition

Sequential composition in this setting resembles functional composition of behaviors. However, we generalize sequential composition by allowing that the output of one component need not be directly fed into the next one.

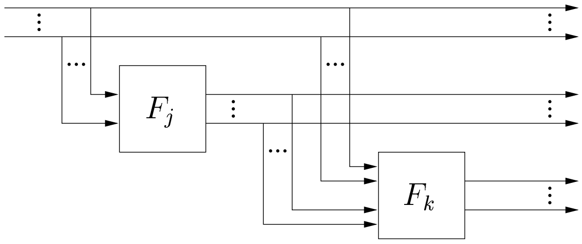

Figure 3.1.: Sequential composition of behaviors.

Given a set of behaviors B_K, the behavior $\otimes B_K$ is called the *sequential composition* of the behaviors in B_K, exactly if:

- K is an ordered set w.r.t. a linear order $<$ such that $(K, <)$ is well-founded, i.e. every non-empty subset of K has a minimal element.
- Any behavior $F_k \in B_K$ only depends on the outputs of preceding behaviors F_j with $j < k$ or on an input of the composed behavior. Formally, $I_k \cap O_l = \emptyset$ for every $l \geq k$.

As no feedback occurs in sequential composition, it is not necessary that any behavior is delayed in order to ensure that $\otimes B_K$ is well-defined. (See the proof in Appendix B.2.)

Strict sequential composition is a specialization of the above, where there is a finite chain of components so that each component's input is the output of its predecessor. Formally, we have the following constraints:

- K is finite. For simplicity it is a set of naturals $\{0, \ldots, n\}$.
- For $k > 0$ it holds that $I_k = O_{k-1}$.

To get strict sequential composition, the intermediate channels are hidden. We define

$$;B_K = (\otimes B_K) \updownarrow_{I_0}^{O_n} .$$

The resulting delay is at least the sum of the delays of the individual behaviors. (The proof is given in Appendix B.2).

3.5. Parallel Composition

Putting a set B_K of behaviors in parallel yields a behavior whose input/output channels consist of all input/output channels of the composed behaviors, $I = \bigcup_{k \in K} I_k$ and $O = \bigcup_{k \in K} O_k$. More precisely, $\otimes B_K$ is called the *parallel composition* of the behaviors in B_K, exactly if the input and output channels of all behaviors are disjoint, i.e. exactly if $\bigcup_{k \in K} I_k \cap \bigcup_{k \in K} O_k = \emptyset$.

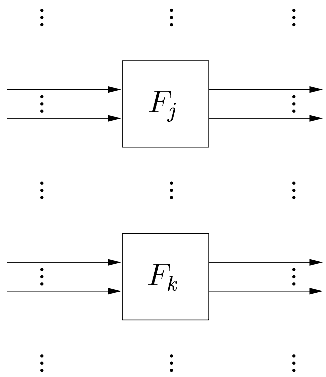

Figure 3.2.: Parallel composition of behaviors.

Note that for parallel composition, as for sequential composition, it is not necessary that any behavior is delayed. The resulting delay is the infimum of the delays of the individual behaviors. (The proofs are given in Appendix B.3.)

3.6. Feedback

The feedback of a behavior $F \in \wp(\overrightarrow{I_F} \xrightarrow{\delta} \overrightarrow{O_F})$ is obtained by connecting its inputs with its compatible outputs.

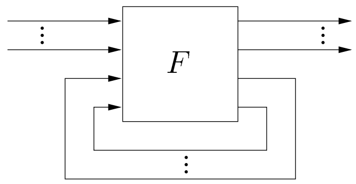

Figure 3.3.: Feedback composition of a behavior.

We obtain the feedback composition $\uparrow F$ of F by applying nested composition to F and hiding the *feedback channels* $C_F = I_F \cap O_F$:

$$\uparrow F = \otimes\{F\} \updownarrow_I^O$$

where $I = I_F \setminus O_F$ and $O = O_F \setminus I_F$. If the set of feedback channels of F is empty, then the original behavior is resulting again.

For the feedback to be well-defined, it is necessary that the behavior F is delayed on the feedback channels. Formally, this means that the restriction of F to the feedback channels $F(x)|_{C_F}$ must be delayed. The delay of $\uparrow F$ then is at least that of F. (The proofs are given in Appendix B.4.)

4. Relevant Specializations of Streams

In the previous sections, we have used a very general kind of streams: Each channel $c \in \mathbb{C}$ was assigned to a dense stream in $M_c^{\mathbb{R}_+}$, which is in general an arbitrary mapping from $\mathbb{R}_+$ into a set of messages M_c. The set M_c can be regarded as type of the data on channel c.

However, not each of these mappings $\overrightarrow{C}$ is of practical interest. Instead, we want some channels to be (piece-wise) continuous, others are signal based and therefore carry sets of signals, and others still are message based, where each message is transmitted in a single time instant.

All these kinds of streams are specializations of the general definition we have given in the previous section. The main purpose of this section is to define an infrastructure for these specializations, and to show important properties, like closedness under certain operations.

In particular, we will define the following three main kinds of streams and examine their properties:

Hybrid Streams are streams that are piece-wise *smooth*. Informally smooth means that there are no abrupt changes in the stream. A formal definition is given later on. Their general form allows arbitrary values at points of discontinuity. However, the points of discontinuity may not be dense in $\mathbb{R}_+$. Often, but not necessarily, the range of a hybrid stream is $\mathbb{R}$, to denote analog values. They are particularly appropriate for modeling physical systems and analog hardware structures [GSB98].

Signal-set streams are streams with a range $\wp(S)$, which is the powerset over signals S. Each signal may occur independently of all others, but must be present for an entire interval $I \subseteq \mathbb{R}_+$, not only a point in time. These streams model discrete hardware structures. Furthermore, they are the basis for many synchronous languages such as Esterel [Ber98] and μ-Charts [Sch98].

Message streams are streams where messages, which are present only for an instant of time, occur sporadically. The time instances at which messages occur may not be dense. Message streams model the transmission of a message over a given channel that happens at a certain point in time. The range of these streams includes the dummy element $\varepsilon \in \mathbb{M}$ to denote the absence of messages. These streams model communication between software components.

Each channel $c \in \mathbb{C}$ therefore has not only a type M_c, but also a *kind* describing to which of these specializations the channel belongs.

4.1. Classification of Streams

For $I \subseteq \mathbb{R}$ and $M \subseteq \mathbb{R}$ a function $f \in I \to M$ is called *smooth* if it is infinitely often differentiable. For $M \not\subseteq \mathbb{R}$, f is called smooth, if it is constant on I[1]. A tuple/set of functions is smooth if all its components/elements are.

Given a single stream $x \in \overrightarrow{c}$, we denote by $D(x) \subseteq \mathbb{R}_+$ the set of all points in time, where stream x is not smooth, i.e. where it exhibits a (higher-order) discontinuity. We may classify x by enforcing:

A1. $D(x)$ may not be dense. In other words each finite interval can be partitioned into finitely many subintervals in such a way that x is smooth on every subinterval. We also say that x is *piecewise smooth.*

A2. Points of (higher-order) discontinuity are equidistant. This means there exists a timing distance $\rho > 0$ so that $D(x)$ consists of multiples of ρ: $D(x) \subseteq \{n\rho | n \in \mathbb{N}\}$

A3. $D(x) = \emptyset$: No (higher-order) discontinuity exists.

Another classification goes along with the question what happens at points $t \in D(x)$ of (higher-order) discontinuity:

B1. The points with discontinuities exhibit arbitrary values.

B2. The values for point t are defined in such a way that x is smooth at time t with respect to an interval to the right, i.e. there is an $\epsilon > 0$ so that x restricted to $[t, t + \epsilon)$ is smooth on this interval.

[1] This corresponds to using the discrete topology on the set M. Variants of this definition based on other topologies for M, and hence on other notions of continuity, are conceivable.

B3. The values for point t are defined in such a way that x is smooth at time t with respect to an interval to the left, i.e. there is an $\epsilon > 0$ so that x restricted to $(t-\epsilon, t]$ is smooth on this interval.

Figure 4.1 shows some examples for these kinds of streams.

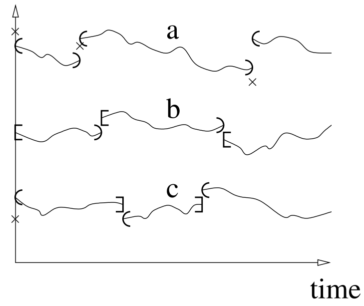

Figure 4.1.: Hybrid streams.

A third criterion for classification results from the question, what is happening within the smooth intervals:

C1. The values may vary arbitrarily.

C2. The values are constant in this interval.

C3. The values are constantly set to the special message $\varepsilon \in \mathbb{M}$ indicating the absence of any message. Correspondingly, only the points of discontinuity may carry messages.

Not all combinations of these classifications are of equal practical interest. In the following section, we will mention just the most interesting ones.

Given a channel $c \in \mathbb{C}$ we denote with $\overset{\ominus\!\!\rightarrow}{c}$ the set of streams that we want to allow on this channel. This set of streams is a subset of all streams: $\overset{\ominus\!\!\rightarrow}{c} \subseteq \overrightarrow{c}$. We regard $\overset{\ominus\!\!\rightarrow}{c}$ to be the *kind* of channel c. We extend this notation to tuples of streams $C \subseteq \mathbb{C}$ by:

$$\overset{\ominus\!\!\rightarrow}{C} = \{\, x \in \overrightarrow{C} \mid \forall c \in C : x(c) \in \overset{\ominus\!\!\rightarrow}{c} \,\}$$

In the following we will use the three enumerations from above to refer to the different kinds of streams. We will write M^{abc} for $a, b, c \in \{1, 2, 3\}$ to refer to the streams in $\mathbb{R}_+ \to M$ that are in kind a of the classification criterion A from above, in kind b of the criterion B and in kind c of the criterion C. For instance, M^{111} denotes the set of piecewise smooth streams $\mathbb{R}_+ \to M$ that may have arbitrary values at the points of discontinuity and that may vary. Sometimes we may also use asterixes to refer to all kinds of the associated classification criterion.

The set of delayed behaviors with at least delay δ for a given $\delta > 0$ and which operate on the kinds of streams presented above, i.e. on M^{***}, is closed with respect to nested composition, parallel composition, sequential composition, feedback, renaming and interface adaption. This is a consequence of the more general theorem in Appendix C.

4.2. Hybrid Streams

Hybrid systems are typically characterized by a non-trivial mixture of discrete and continuous aspects. Examples of hybrid systems are embedded real-time controllers in their physical environment or the environment itself. Due to the increasing importance of embedded real-time systems, hybrid systems are an increasingly active area of research.

In our view, the interface behavior of hybrid system components is determined by time periods in which no discrete actions occur, i.e. the values at the interface only change smoothly, and by time instances at which discrete actions take place and (possibly) cause discontinuities in a component's output.

A realistic component cannot perform discrete actions at arbitrary frequency as it is not infinitely fast. Hence, we demand that a stream x in a hybrid system only has finitely many discontinuities during any finite interval $I \subset \mathbb{R}_+$. Correspondingly, we are interested in the piecewise smooth streams here. In the context of hybrid systems we often refer to piecewise smooth streams as *hybrid streams*.

Example 2 (Three kinds of hybrid streams) Figure 4.1 shows three particular examples of hybrid streams.

Kind M^{11*}. For stream a, every finite open interval can be partitioned into finitely many open intervals and a finite set $\mathcal{T}$ containing the intervals' boundary points so that a is smooth on each interval.

Kind M^{12*}. For stream b, every finite leftclosed, rightopen interval can be partitioned into finitely many leftclosed, rightopen intervals so that b is smooth on each interval.

Streams of this class are the basis for the model in [GSB98]. In the automata-based description technique developed there, discrete actions correspond to transitions in automata. This particular partitioning of the streams is selected, because transitions in this model are taken instantaneously when they become enabled (therefore the intervals in the partitioning are rightopen) and determine the start values for the next period of smooth evolution (therefore the intervals are leftclosed).

Kind M^{13*}. For stream c, every finite leftopen, rightclosed interval can be partitioned into finitely many leftopen, rightclosed intervals so that c is smooth on each interval.

The demand that the output of behaviors may not depend on future inputs leads to a fundamental difference between the above kinds of streams. Assume a behavior with delay $\delta > 0$ to be given. For kind M^{11*} and M^{12*} the behavior can use the limit from the left $lim_{x\uparrow t}$ of an input stream to get the expected value of the stream at time t. By comparing this value with the actual value at time t it can detect discrete changes, or *events*, in the input and react to them at time $(t + \delta)$. If, however, the input streams are smooth on rightclosed intervals, like for kind M^{13*}, the limit from the left does not permit the detection of changes. The limit from the right $lim_{x\downarrow t}$ may not be used, as it would influence the output in time $(t + \delta)$ and therefore violate the delay assumption. Note that for kind M^{11*}, a possible discrete change from time t to the time just after t also cannot be detected because of the same reason. $\square$

4.3. Signal-set Streams

Communication using signal-set streams can mainly be found in discrete hardware structures. To allow the detection of discrete signals they must be present for entire time intervals.

For signal-set streams the powerset over a set of signals $\mathcal{S}$ is used as the range of the streams, i.e. $M = \wp(\mathcal{S})$. Each signal may occur independently of all others, but it must be present for an entire time interval, not only for one instance. Therefore, piecewise constant streams with equidistant points of discontinuity can be used to model this kind of streams. These streams are of kind M^{2*2}. It is to some extent arbitrary what the streams do on points of discontinuity, but using kind M^{222} is certainly a good choice.

For asynchronous hardware structures it may furthermore be appropriate to allow points of discontinuity that are not equidistant and use streams of kind M^{1*2} instead of M^{2*2}.

4.4. Message Streams

Message based communication is mainly useful for the software part of a system, but can equally well be used to model business enterprise processes, where not information but letters, or even goods flow on the channels. Furthermore, message streams can be used as a discrete event abstraction of continuous processes.

In this model, we assume messages to be transmitted in one instant of time. Based on the above given classification, we regard a channel $c \in \mathbb{C}$ to be message based if its streams are of the kind M^{113} with $\varepsilon \in M$.

As messages are produced and processed in a discrete manner, we only allow a finite number of messages during any finite time interval. Assuming the production of a message takes some minimum amount of time, permitting infinitely many messages in a finite interval would require the producing unit to work faster than this minimum amount.

In Section 5.2 we will transform these dense message streams into timed and even untimed message streams. They allow to disregard detailed timing, if (or: as long as) we do not want to talk about it.

4.5. Kind Respecting Behaviors

Having now a more accurate definition of what kinds of streams are of real interest, the definition of stream processing functions must be revisited. In particular, we want to identify those behaviors that always operate on the same kinds of streams, i.e. respect the type of their channels.

Definition 3 (Kind respecting functions and behaviors) *For a function* $f \in \overrightarrow{I} \overset{\delta}{\to} \overrightarrow{O}$ *with a delay* $\delta \geq 0$ *we say* f respects the kind of all it channels*, if for all streams* $x \in \overset{\ominus\!\!\twoheadrightarrow}{I}$ *it holds that* $f(x) \in \overset{\ominus\!\!\twoheadrightarrow}{O}$*. A behavior* respects the kind of all its channels *if all its contained functions do. As we disregard the output upon inputs in* $\overrightarrow{I} \setminus \overset{\ominus\!\!\twoheadrightarrow}{I}$*, we restrict kind respecting behaviors to inputs in* $\overset{\ominus\!\!\twoheadrightarrow}{I}$ *and therefore describe these behaviors by the set* $\overset{\ominus\!\!\twoheadrightarrow}{I} \overset{\delta}{\to} \overset{\ominus\!\!\twoheadrightarrow}{O}$*, which describes* component kinds*.*

Note that although $\overrightarrow{I} \overset{\delta}{\to} \overrightarrow{O}$ and $\overset{\ominus\!\!\twoheadrightarrow}{I} \overset{\delta}{\to} \overset{\ominus\!\!\twoheadrightarrow}{O}$ do have different domains and ranges, each function in $\overset{\ominus\!\!\twoheadrightarrow}{I} \overset{\delta}{\to} \overset{\ominus\!\!\twoheadrightarrow}{O}$ can easily be extended to a function in $\overrightarrow{I} \overset{\delta}{\to} \overrightarrow{O}$.

One important property of the composition operators defined in Chapter 3 is that if a set of kind respecting behaviors is composed, then the result will also be of the appropriate kind. In more detail, the set of delayed kind respecting behaviors (and functions) which at least have delay $\delta > 0$ for a given δ is closed with respect to nested composition, sequential composition, parallel composition, feedback, renaming and interface adaption. The proof is given in Appendix C. Note that renaming of channels now must not only respect the type of the underlying messages M_c but also the kind of the channels.

5. Discrete Layer of Time

For some kinds of dense streams $x \in \overset{\ominus\!\!\rightarrow}{c}$ we can define abstractions that allow us to use the work that has been done for discrete streams.

Before we define the connection between the discrete and the continuous stream layers, we will introduce the discrete stream theory briefly.

5.1. Definitions

As the definitions for discrete streams are to a large extent very similar to the already given ones, we just mention the existing operations and differences to the previous ones.

An *untimed stream* is a finite or infinite sequence of messages. If M denotes a set of messages, M^* denotes the set of all finite sequences of messages and M^∞ the set of all infinite sequences of messages, for the set of all streams over M, denoted by M^ω, we define:

$$M^\omega = M^\infty \cup M^*$$

We will use the following operations on streams:

- $. \frown . : M^\omega \times M^\omega \to M^\omega$ denotes the concatenation of two streams, i.e. the stream which is obtained by putting the second argument after the first. $\frown$ will also be used to concatenate a single message with a stream.
- $\# : M^\omega \to \mathbb{N} \cup \{\infty\}$ gives the length of a stream as a natural number or ∞, if the stream is infinite.
- $© : \wp(M) \times M^\omega \to M^\omega$ denotes the filter-function. $©(N, s)$ deletes all elements in s which are not contained in set N.

Let us denote by $\overline{c}$ the set of untimed streams over M_c. Formally such streams are in M_c^ω, denoting an infinite or finite sequence of messages.

Let us furthermore denote by $\tilde{c}$ the set of (discrete) *timed streams* over M_c. Formally such streams are in $(M_c^*)^\infty$, denoting an infinite sequence of finite intervals, where each interval contains the finite sequence of messages that occurred in that interval of time. All time intervals are of the same length.

For every timed stream x we abbreviate the selection of the n-th time interval by $x(n)$, where the first interval is selected with index 1. For every timed stream x we abbreviate the restriction to the first n intervals by $x{\downarrow}n$.

For every untimed stream x we abbreviate the selection of the n-th message (if it exists) by $x(n)$. For every untimed stream x we abbreviate the restriction to the first n elements by $x{\downarrow}n$. As before, we carry this operations over to named stream tuples.

For named stream tuples we use the same restriction and combination operators, $|$ and $+$, as before. Furthermore we extend these operators and the above selections to sets of streams in a pointwise style.

Timed stream processing functions are denoted by $f : \tilde{I} \overset{n}{\to} \tilde{O}$ where the natural number $n \geq 0$ tells us about the delay of the function. It holds that:

$$\forall x, y \in \tilde{I}, k \in \mathbb{N} : x{\downarrow}k = y{\downarrow}k \;\Rightarrow\; f(x){\downarrow}(k+n) = f(y){\downarrow}(k+n)$$

Again each timed function f can be attached the maximal delay time that it has by $\delta(f)$, which is now a natural number.

For untimed stream processing functions there is no notion of delay, they just have to be continuous in the sense of Scott's domain theory [SG90, Win93]. In our terminology, the set of untimed stream processing functions is denoted by $\overline{I} \overset{utd}{\to} \overline{O}$.

Again we extend the definitions to sets of functions in the same way we used in Section 2.6 for sets of (dense) stream processing functions.

5.2. Transformation Between Dense, Timed and Untimed Behaviors

Timed streams are an abstraction of the continuous message based streams. Untimed streams in turn are an abstraction of timed streams.

Therefore, we can define an operator ∇ to transform dense streams to (discrete) timed ones and to transform (discrete) timed streams to untimed streams. Some variants of ∇ are given in the following sections. Others are possible.

For example, in [Bro97] a version of ∇ is defined that transforms streams of kind M^{122}, i.e. streams that are piecewise constant and right-continuous, to an equivalent of message streams.

5.2.1. From Message Streams to Timed Streams

When abstracting from a dense stream $M^{\mathbb{R}_+}$ to a discrete timed stream of kind $(M^*)^\infty$ a *duration of the time unit* $T > 0$ has to be introduced. This duration T for example corresponds to a global clock ticking in a system, and can also be seen as sampling period.

For message streams M^{113} (Section 4.4) we have a special $\varepsilon \in M$ denoting the absence of a message and the ∇ operator with time unit duration $T > 0$ is defined as follows:

$$\begin{array}{lcl} \nabla_T & \in & M^{113} \rightarrow (M^*)^\infty \\ \nabla_T(x) & = & \{y \in (M^*)^\infty \mid \\ & & \quad \forall n \in \mathbb{N}.\ y(n) = \phi_{[(n-1)\cdot T,(n\cdot T))}(x)\}, \end{array}$$

where $\phi_I(x)$ extracts all messages from the interval I in x into a sequence that preserves the ordering. Formally:

$$\begin{array}{lcl} \forall t \in I : x(t) = \varepsilon & \Rightarrow & \phi_I(x) = \epsilon\ \wedge \\ \exists t.\ t = min\{t \in I | x(t) \neq \varepsilon\} & \Rightarrow & \phi_I(x) = x(t) \frown \phi_{I \cap (t,\infty)}(x), \end{array}$$

where ϵ for timed streams denotes the empty list. The minimum exists, because we did only allow finitely many discontinuities within any finite interval.

If the distance between any two messages is greater than the duration of the time unit T, then the resulting timed stream is furthermore an element of M^∞, with $\varepsilon \in M$ denoting the absence of any message in a time interval.

5.2.2. From Step Streams to Timed Streams

The notion *step streams* refers to streams of kind M^{222}, i.e. to dense streams where discontinuities may only occur at times $n \cdot T$ for $n \in \mathbb{N}$ and a constant $T > 0$. Between those time points the streams are right-continuous and constant.

For stream $x \in M^{222}$ with discontinuities at $n \cdot T$, ∇_T is defined as follows:

$$\begin{array}{l} \nabla_T \ \in\ M^{222} \rightarrow (M^*)^\infty \\ \forall n \in \mathbb{N}.\ (\nabla_T(x))(n) = [x(n \cdot T)] \end{array}$$

where $[x(n \cdot T)]$ denotes a one-element list with element $x(n \cdot T)$. Note that a new message occurs in the timed stream even if the value of the dense stream does not change between consecutive intervals.

5.2.3. From Timed to Untimed Streams

The abstraction $\nabla(x)$ from timed to untimed streams results from concatenating all sequences in stream x [Bro97]. It is inductively defined as follows:

$$
\begin{aligned}
&\nabla \in (M^*)^\infty \to M^\omega \\
&\nabla(a \frown x) = a \frown \nabla(x)
\end{aligned}
$$

for $a \in M^*$ and $x \in (M^*)^\infty$.

5.2.4. Abstraction of Behaviors

Based on the abstraction operators $\nabla_T : \mathbb{M}^{***} \to (\mathbb{M}^*)^\infty$ for streams from above, abstraction ∇_T can be extended to sets of stream processing functions. The extended operator abstracts from dense stream processing functions to (discrete) timed stream processing functions.

It is quite common that abstraction results in a loss of information. In the context of functions the abstraction of the input streams leads to a loss of information the function can rely on. This loss becomes apparent, when e.g. two dense streams x and y with $\nabla_T(x) = \nabla_T(y)$ but $\nabla_T(f(x)) \neq \nabla_T(f(y))$ are considered. In this case a naive abstraction $\nabla_T(f)$ of function f defined by $\forall x.\ \nabla_T(f)(\nabla_T(x)) = \nabla_T(f(x))$ is contradictory, no such function $\nabla_T(f)$ exists.

Therefore, we choose a set based approach which allows the abstraction from dense to discrete timed functions to introduce underspecification. In order to ensure that the discrete timed functions again are behaviors, each function $g \in \nabla_T(f)$ is defined in dependency of a set $X \subseteq \overset{\ominus\!\!\rightarrow}{I}$ which is the inverse image of the set of timed streams $\tilde{I}$ and guarantees that streams in $\tilde{I}$ with equal prefixes have inverse images in X with equal prefixes of corresponding length. Formally, we define:

$$
\begin{aligned}
&\nabla_T : (\wp(\overset{\ominus\!\!\rightarrow}{I} \xrightarrow{\delta} \overset{\ominus\!\!\rightarrow}{O})) \to (\wp(\tilde{I} \xrightarrow{n} \tilde{O})) \\
&g \in \nabla_T(F) \Leftrightarrow \exists f \in F.\ \exists X \in \mathcal{T}_{\nabla_T}(\overset{\ominus\!\!\rightarrow}{I}).\ \forall x \in X.\ g(\nabla_T(x)) = \nabla_T(f(x))
\end{aligned}
$$

where selecting a subset $X \in \mathcal{T}_{\nabla_T}(\overset{\ominus\!\!\rightarrow}{I})$ guarantees that ∇_T is bijective on $X \to \tilde{I}$. This ensures that for a given X and given $f \in F$ the resulting function g is uniquely determined on each input, but not overspecified:

$$
\begin{aligned}
\mathcal{T}_{\nabla_T}(\overset{\ominus\!\!\rightarrow}{I}) = \{\ X \subseteq \overset{\ominus\!\!\rightarrow}{I} \mid\ & \nabla_T(X) = \tilde{I}\ \wedge \\
& \forall x, y \in X.\ \forall n.\ \nabla_T(x){\downarrow}n = \nabla_T(y){\downarrow}n \Rightarrow \\
& \qquad x{\downarrow}(n \cdot T) = y{\downarrow}(n \cdot T)\ \}
\end{aligned}
$$

For the ∇_T operators from above we can easily prove that $\mathcal{T}_{\nabla_T}(\overset{\ominus\!\!\!\rightarrow}{I})$ is non-empty and that $\nabla_T(F)$ is non-empty provided F is non-empty (Appendix D).

Depending on the concrete choice of ∇_T and T the exact delay of the functions in F need not be preserved in the abstraction. However, the delay n of the discrete time functions can be chosen as the largest natural number with $n \cdot T \leq \delta$. (The proof is given in Appendix D.) In the worst case, only $n = 0$ is valid. This happens, if the chosen sampling time T is greater than the smallest occurring delay δ.

From a methodical point of view it is also interesting to have the converse operation, where the abstract behavior G is given, and a translation into the set of dense stream processing functions is desired. We define this converse operation as $\nabla_T^{-1}(G) = \{f \mid \exists g \in G.\ g \in \nabla_T(\{f\})\}$. As we can see from the definitions, the operation is monotonic with respect to set inclusion. We can therefore use the already well established refinement and composition techniques on discrete timed streams as long as the continuous parts are not considered. When the continuous parts become important, a transformation with ∇_T^{-1} is possible and still all refinement arguments are valid.

$$\begin{tikzcd}
G \arrow[r, "\supseteq"] \arrow[d, "\nabla_T^{-1}"'] & G' \arrow[d, "\nabla_T^{-1}"] \\
F \arrow[r, "\supseteq"'] & F'
\end{tikzcd}$$

Figure 5.1.: Reasoning with different time models.

In other words, we can work with the abstract and more easily tractable discrete time system G using refinement and composition techniques on discrete timed streams e.g. to obtain $G' \subseteq G$. Using the inverse relation we can derive $F = \nabla_T^{-1}(G)$ and $F' = \nabla_T^{-1}(G')$ with $F' \subseteq F$, thus translating the abstract, discrete derivation into the theory of dense steams. Figure 5.1 depicts this method as a commuting diagram.

6. An Example

As an example for the usage of the stream kinds and time models presented in the previous chapters we consider an abstract model of the controller of a production cell (Fig. 6.1). The controller operates on different kinds of streams and we will employ different time models in order to model its communication to other components on an appropriate level of abstraction.

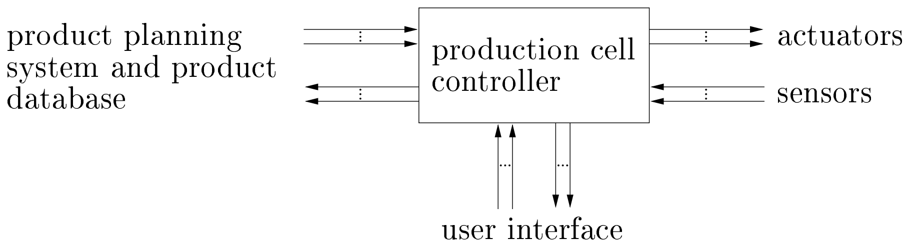

Figure 6.1.: The production cell's controller.

On the one hand the controller communicates with a product planning system and a product database. From the product planning system it receives orders and from the product database it gets the information telling it how to fulfill the order. After a product is finished information about the product's quality and manufacturing costs is sent to the responsible product planning system. The product planning system is essentially a business information system and therefore the communication to it does not rely heavily on real-time communication. Hence, this kind of communication is a good candidate for channels with untimed message streams.

Furthermore, the controller communicates with actuators and sensors of the production cell. This communication is partly event based, telling for example the arrival of certain goods at certain places. However, time plays a certain role here. This part is modeled best as communication over channels with timed message streams.

Another part of the communication involves the continuous transmission of analog information. For example, sensors that permanently measure the present temperature in an oven of the production cell produce such information. The controller is obliged to continuously adjust the amount of energy emitted by the heating inside the oven so that a certain temperature trajectory is obtained in the oven. In an implementation, this is typically discretized using appropriate sampling techniques. However, for a treatment in a specification, it is more natural to handle this by analog streams of kinds M^{1*1}.

Furthermore, the controller is obliged to communicate its current operational mode and status to a primitive user interface, which merely consists of a set of lamps. It continuously sends sets of signals to this interface in order to light appropriate lamps. These signals can conveniently be modeled as signal set streams.

The controller is a good example for a component with different kinds of communication channels. In principle all communication could have been handled using the basic kind of streams we defined at first. However, it is much more convenient to use different kinds of streams for different purposes. For example, it is not necessary to work in a dense time model when elaborating the interaction of the controller with business information systems. Here an untimed model usually suffices.

The abstraction operators from Section 5.2 support this methodology. The idea behind them is to allow system development in the time model which is most appropriate for the problem at hand and switch, when appropriate.

7. Conclusion

Based on streams and stream processing functions we introduced an abstract mathematical model in this paper that describes dense nondeterministic behavior on dense input and output streams and that is suitable for integrating discrete and continuous system views.

Motivated by time critical application areas, like hybrid systems and discrete hardware structures, and by areas like information systems where timing is not critical we have identified several important classes of *hybrid* streams.

Furthermore, we presented a set of very general composition operators. A nested composition operator allows to compose arbitrary, including uncountable infinite, sets of components that may communicate over very general, even uncountable infinite, sets of channels.

Please note that it was not the purpose of this paper to introduce a practical specification language. Instead it aims at providing a very general, sound mathematical model that can serve as semantic foundation for a variety of specification languages.

A. Mathematical Treatment of Dense Streams

A mathematical treatment of functional specifications requires dealing with feedback loops. In the discrete case the semantics of such loops has been successfully described as least fixed points of functions over domains [Bro93, BDD$^+$93]. The underlying mathematical model is Scott's domain theory [SG90, Win93]. Fixed points of stream processing functions over dense streams, however, are more naturally and elegantly described by the fixed point theory of Banach.

In order to specify feedback loops of stream processing functions in Chapter 3, we therefore introduce the main concepts of metric space theory.

A.1. The Metric Space of Dense Streams

Definition 4 (Metric Space) *A* metric space *is a pair (D, d) consisting of a nonempty set D and a mapping $d : D \times D \to \mathbb{R}_+$, called a* metric *or a* distance, *which has the following properties:*

(1) $\forall x, y \in D : \quad d(x, y) = 0 \quad \Leftrightarrow \quad x = y$
(2) $\forall x, y \in D : \quad d(x, y) = d(y, x)$
(3) $\forall x, y, z \in D : \quad d(x, y) \leq d(x, z) + d(z, y)$ *(triangle inequality)*

For dense streams, we use the following metric:

Definition 5 (The Metric of Streams) *The metric space of dense streams $(M^{\mathbb{R}_+}, d)$ is for all $x, y \in M^{\mathbb{R}_+}$, $M \neq \emptyset$, defined as follows:*

$$d(x, y) = inf\{2^{-t} \mid t \in \mathbb{R}_+ \wedge x{\downarrow}t = y{\downarrow}t\}$$

It is easy to prove that d is indeed a metric. It closely resembles the Baire metric defined in [Eng77], but is not equivalent to it.

Please note, that we do not use any implicit structure of the set of messages M, instead we assume M to be a flat set of messages (in the words of the domain theory [SG90]). We use the same metric for the subspaces M^{***} of $M^{\mathbb{R}_+}$ which were defined in Section 4.1.

From this definition of the Baire metric for single streams a metric d_C for named tuples of streams $\overrightarrow{C}$ can be easily derived. For a set C of channel names and named stream tuples $x, y \in \overrightarrow{C}$ we define d_C as:

$$d_C(x, y) = sup\{d(x(c), y(c)) \mid c \in C\}$$

Theorem 1 *d_C is a metric on $\overrightarrow{C}$.*

Proof 1 d_C is well-defined, because for every $c \in C$ and $x, y \in \overrightarrow{C}$ value $d(x(c), y(c))$ is bounded above by 1. With using the definition of *sup*, the rest of the proof is straightforward. $\square$

This metric definition works for arbitrary, including uncountable infinite, channel sets C.

We call $(\overrightarrow{C}, d_C)$ the *metric space of named stream tuples.* In the following we will only write d instead of d_C when the context makes clear that we are operating on named stream tuples.

Obviously, d_C is also a metric on the named stream tuples with channel kinds $\overset{\ominus\!\!\rightarrow}{C}$. We therefore call $(\overset{\ominus\!\!\rightarrow}{C}, d_C)$ the *metric space of named stream tuples with channel kinds.*

A.2. Completeness

In order to derive the desired results, namely the unique existence of fixed points for feedback loops, we need to introduce the notion of sequences and their limits.

Definition 6 (Convergence) *A sequence of elements (a_i) in a metric space (D, d) is* converging *to $a \in D$, if for every $\epsilon > 0$ there exists a number $k \in \mathbb{N}$ such that $d(a_i, a) \leq \epsilon$ for all $i \geq k$ [Eng77].*

A sequence of streams, for example, converges to a certain stream, if the coincident prefixes become increasingly larger. The following yields a helpful technique for proving the convergence of a sequence.

Definition 7 (Cauchy Sequence) *For a metric space (D, d) a sequence of elements (a_i) in D is called a* Cauchy sequence*, if for every $\epsilon > 0$ there exists a number $k \in \mathbb{N}$ such that $d(a_i, a_k) \leq \epsilon$ for all $i \geq k$ [Eng77].*

Definition 8 (Complete Metric Space) *A metric space (D, d) is called* complete*, if every Cauchy sequence in D converges to an element in D [Eng77].*

Theorem 2 *The metric spaces $(M^{\mathbb{R}_+}, d)$ and (M^{***}, d) are complete.*

Proof 2 Let (a_i) be a Cauchy sequence in one of the above spaces.

We define the fixed point a of (a_i) as follows: For each $t \in \mathbb{R}_+$ we choose a k such that $\forall i \geq k.\, a_i \downarrow t = a_k \downarrow t$ and define $a \downarrow t = a_k \downarrow t$. Such a k exists, because of the definition of the Baire metric and of Cauchy sequences. Clearly, the sequence (a_i) converges to a.

As a is a function from $\mathbb{R}_+$ to M we immediately conclude $a \in \mathbb{R}_+ \to M = M^{\mathbb{R}_+}$.

The kinds M^{ijk} for $i, j, k \in \{1, 2, 3\}$ of dense streams are all characterized by some property that, if falsified by a stream, is already falsified by a finite prefix of it (this can be easily verified). Therefore, if a is not in M^{ijk} there must be a time t where a fails to satisfy the characterizing property. Due to construction this implies that some of the (a_i) must already have failed being in M^{ijk}. Thus all M^{ijk} are closed. □

Theorem 3 *For arbitrary $C \subseteq \mathbb{C}$ the metric space of named stream tuples $\overrightarrow{C}$ is complete. The limit a of a Cauchy sequence (a_i) is given by pointwise application: $a(c) = lim_{n \to \infty} a_n(c)$.*

Proof 3 Let (a_i) be a Cauchy sequence in $\overrightarrow{C}$. Then the component sequences $(a_i(c))$ in $\mathbb{M}_c^{\mathbb{R}_+}$ are Cauchy sequences, too. Let $a \in \overrightarrow{C}$ be the function resulting from pointwise construction. It exists, because every $\mathbb{M}_c^{\mathbb{R}_+}$ is a complete metric space.

The Cauchy definition gives us:

$$\forall \epsilon > 0.\ \exists k \in \mathbb{N}.\ \forall i \geq k.\ d(a_i, a_k) \leq \epsilon$$

Expanding the definition of d, we get:

$$\forall \epsilon > 0.\ \exists k \in \mathbb{N}.\ \forall i \geq k, c \in C.\ d_c(a_i(c), a_k(c)) \leq \epsilon$$

Note that this is stronger than having a Cauchy sequence on each channel, because the k only depends on ϵ and is guaranteed to exist universally for all channels.

As for each channel c the distance between $a_i(c)$ and $a_k(c)$ is limited by ϵ, so is the distance between $a_k(c)$ and $a(c)$, which can easily be shown using that every $\mathbb{M}_c^{\mathbb{R}_+}$ is a complete metric space. Therefore:

$$\forall \epsilon > 0.\ \exists k \in \mathbb{N}.\ \forall c \in C.\ d_c(a_k(c), a(c)) \leq \epsilon$$

By definition of d as supremum over the channels, we derive:

$$\forall \epsilon > 0.\ \exists k \in \mathbb{N}.\ d_C(a_k, a) \leq \epsilon$$

and therefore (a_k) converges to a. □

Note that the proof is not restricted to finite or countable sets of channels, but C may have arbitrary size.

A similar proof shows that the metric space of named streams with channel kinds $\overset{\twoheadrightarrow}{C}$ is complete.

A.3. Contractive Functions and Fixed Points

Before reasoning about the existence of fixed point solutions for composition with feedback, we need to introduce the notion of Lipschitz functions and contractive functions.

Definition 9 *(Lipschitz functions) Let (D_1, d_1) and (D_2, d_2) be metric spaces and let $f \in D_1 \to D_2$ be a function. We call f* Lipschitz function *with constant $c \geq 0$ if the following condition is satisfied:*

$$d_2(f(x), f(y)) \leq c \cdot d_1(x, y)$$

If $c = 1$ we call f non-expansive*. If $c < 1$ we call f* contractive.

Theorem 4 *The composition of two Lipschitz functions $f \in D_1 \to D_2$ and $g \in D_2 \to D_3$ with constants c_1 and c_2 is a Lipschitz function with constant $c_1 \cdot c_2$.*

Proof 4 $d(g(f(x_1)), g(f(x_2)) \leq c_2 \cdot d(f(x_1), f(x_2)) \leq c_2 \cdot c_1 \cdot d(x_1, x_2)$ □

Corollary 1 *The composition of a contractive and a non-expansive function is contractive. The composition of two non-expansive functions is non-expansive. Identity is non-expansive.*

The main tool for handling recursion in metric spaces is Banach's fixed point theorem. It guarantees the existence of a unique fixed point for every contractive function.

Theorem 5 *(Banach's fixed point theorem) Let (D, d) be a complete metric space and $f \in D \to D$ a contractive function. Then there exists an $x \in D$, such that the following holds:*

$$\begin{array}{lll} \textbf{(1)} & x = f(x) & \textit{(x is a fixed point of f)} \\ \textbf{(2)} & \forall y \in D: \; y = f(y) \Rightarrow y = x & \textit{(x is unique)} \\ \textbf{(3)} & \forall z \in D: \; x = lim_{n \to \infty} f^n(z) & \textit{where} \\ & \quad f^0(z) \quad = z & \\ & \quad f^{n+1}(z) \; = f(f^n(z)) & \end{array}$$

Proof 5 See [Eng77] or [Sut75]. □

Usually we want to use a parameterized version of this theorem.

Definition 10 *(Parameterized fixed point) Let $f \in D \times \hat{D} \to D$ be a function of complete metric spaces that is contractive in its first argument. We define the* parameterized fixed point *function μf as follows:*

$$(\mu f) \in \hat{D} \to D$$
$$(\mu f)(\hat{y}) = x$$

where x is the unique element of D such that $x = f(x, \hat{y})$ as guaranteed by Banach's fixed point theorem.

Theorem 6 *If f is contractive with constant c so is μf.*

Proof 6 The theorem follows immediately from [MPS86] pages 114–115. □

Please note that μf may even have a contractivity higher that f, but the original contractivity c is at least guaranteed.

A.4. Delayed Behaviors and Contractivity

Before we come to the composition operators we need to establish the connection between (delayed) deterministic behaviors and contractive functions.

Theorem 7 *A stream processing function is delayed with delay $\delta \geq 0$ exactly if it is a Lipschitz function with constant $1 \geq c = 2^{-\delta} \geq 0$ with respect to the metric space of named stream tuples. Therefore, it is contractive ($c < 1$) exactly if $\delta > 0$.*

Proof 7 First, we prove the only-if-direction. Assume f is delayed with delay $\delta \geq 0$ and two stream tuples x and y are given which differ from time point t_0. Therefore $d(x,y) = 2^{-t_0}$. It holds that $x \downarrow t_0 = y \downarrow t_0$ which by delay of f leads to $f(x) \downarrow t_0 + \delta = f(y) \downarrow t_0 + \delta$. By definition of d, we get $d(f(x), f(y)) \geq 2^{-t_0-\delta} = 2^{-\delta} \cdot d(x,y)$. Hence, f is a Lipschitz function with constant $c = 2^{-\delta}$. For deterministic behaviors without delay, i.e. for $\delta = 0$ we get $c = 1$.

Now, we prove the if-direction. Assume f is a Lipschitz function with constant $1 \geq c \geq 0$ and two stream tuples x and y are given which differ from time point t_0. Again, $d(x,y) = 2^{-t_0}$ which implies that $x \downarrow t_0 = y \downarrow t_0$. From the Lipschitz condition it follows that $d(f(x), f(y)) \leq c \cdot d(x,y)$, which implies $f(x) \downarrow t_1 = f(y) \downarrow t_1$ for $t_1 = -ld(c \cdot d(x,y)) = -ld\ c - t_0$. Hence f is delayed with delay $\delta = t_1 - t_0 = -ld\ c > 0$. (For $c = 0$, we take $ld\ c = -\infty$ and get $\delta = \infty$.) For $c = 1$ we get $\delta = 0$, i.e. f is a deterministic behavior without delay. $\square$

Corollary 2 *A stream processing function is non-expansive exactly if it is delayed with delay $\delta = 0$. It is contractive exactly if it is delayed with delay $\delta > 0$.*

B. Proofs About the Composition Operators

In the following proofs we will only regard the metric space of named stream tuples. Nevertheless, the proofs also hold for the other kinds of streams M^{***}, for tuples consisting of different kinds of streams and also for named stream tuples and named stream tuples with channel kinds, as all we need here are complete metric spaces.

B.1. Nested Composition

Theorem 8 *If all behaviors in B_K are delayed with $\delta = inf\{\delta(F_k)|k \in K\} > 0$, then $\otimes B_K \in \wp(\overrightarrow{I} \to \overrightarrow{O})$ is well-defined, where $O = \bigcup_{k \in K} O_k$ and $I = (\bigcup_{k \in K} I_k) \setminus O$.*

Proof 8 Let $In = \bigcup_{k \in K} I_k$. By definition of $\otimes B_K$ each function of the result exhibits correct behavior. We therefore need to prove that there is at least one deterministic behavior f in $\otimes B_K$.

For this, we select a deterministic behavior $f_k \in F_k$ for each $k \in K$ and define a new function $g \in \overrightarrow{In} \to \overrightarrow{O}$ with the unique characterization as follows:

$$\forall k \in K : g(i)|_{O_k} = f_k(i|_{I_k})$$

This function can be seen as a relative of $\otimes B_K$ where no outputs are fed back to inputs. The delay of g is $\delta(g) \geq inf\{\delta(f_k)|k \in K\}$ which is greater than 0 because of the assumption.

We now adapt the interface of g by extending its input interface to obtain $g' \in \overrightarrow{O} \times \overrightarrow{I} \to \overrightarrow{O}$. It is defined by $g'(x, i) = g((x+i)|_{In})$, i.e. inputs on $O \setminus In$ are now fed in, but ignored in the behavior. Hence, g' is also delayed with $\delta(g') = \delta(g) > 0$.

Finally, we define function $f \in \overrightarrow{I} \to \overrightarrow{O}$ to be the parameterized fixed point of g', $f = \mu g'$. Since g' is delayed and hence contractive, f is well-defined. Furthermore it is delayed, because of Theorem 6. By the construction of f and the definition of $\otimes B_K$ one can easily show that $f \in \otimes B_K$. □

Note that by construction of f in the proof, it follows that any combination of deterministic behaviors f_k contributes to the resulting composition. Therefore, the composition of B_K does not impose any assumption on the behavior of its composed elements. Especially it does not restrict the behavior of any component to conform subset behavior. This is in contrast to typical relational and trace composition operators, where through composition a component's behavior is synchronized and therefore implicitly restricted. Therefore, refinement of the components independently of their context is possible, which gives much greater modularity.

Theorem 9 *If all behaviors in B_K are delayed with $inf\{\delta(F_k)|k \in K\} > 0$, then $\otimes B_K$ is delayed with $\delta(\otimes B_K) \geq inf\{\delta(F_k)|k \in K\}$.*

Proof 9 From the construction of $\otimes B_K$ it follows that each $f \in \otimes B_K$ is constructed using deterministic component behaviors $f_k \in F_k$. As $\delta(f_k) \geq \delta(F_k)$ the parallel construction yielding g resp. g' like in Proof 8 also exhibits $\delta(g') = \delta(g) \geq inf\{\delta(f_k)|k \in K\}$. By Theorem 6 with $f = \mu g'$ we get:

$$\delta(f) \geq \delta(g') \geq inf\{\delta(f_k)|k \in K\}$$

Using this, the theorem follows in a straightforward way. □

Actually the existence of a lower bound $inf\{\delta(F_k)|k \in K\} > 0$ is not really necessary for a well-defined nested composition. Instead it can be weakened to the constraint, that each F_k needs to be delayed, without having a global lower bound. In this case the composition is well-defined, but the result may not be delayed with $\delta > 0$ anymore if infinitely many components are involved. However, to prove this the given proof cannot be used, as the construction of the fixed point needs to cope with individual delays for each component.

Also note that even this constraint is still not the most general one. As the construction of g and g' shows, it is only necessary to have a delay $\delta > 0$ in each feedback loop. If individual delays exist for each input/output channel-pair (i, o) in each feedback loop, the total delay in the loop is given by the sum of the individual delays $\sum \delta(i, o)$. To guarantee well-definedness at least one delay $\delta(i, o) > 0$ must be involved. In case of infinitely many components to be composed, or components with infinitely many channels, no global delay $\delta > 0$ must exist.

However, this still is not the most general approach, as it is sufficient to demand delays for different input/output channel-pairs (i, o) of a feedback loop at different points of time. It then suffices that at each point of time t, there is one input/output channel-pair in the loop with a delay $\delta(t, i, o) > 0$. For this kind of restriction, δ should be smooth in its argument t and $t + \delta(t, i, o)$ should never decrease, as a decrease would allow that the future predicts the past. This requirement can be expressed as $\frac{d}{dt}\delta(t,i,o) \geq -1$.

All these approaches are rather sophisticated and unless hard real-time constraints with non delayed components are involved, it usually suffices to restrict to delayed behaviors with appropriate delay $\delta > 0$.

Theorem 10 *Let b_K be delayed deterministic stream processing functions with $inf\{\delta(b_k)|k \in K\} > 0$. Then $\otimes b_K$ again is a deterministic stream processing function.*

Proof 10 We need to prove that $\otimes\{b_K\}$ contains exactly one element. Revisiting the proof for the well-definedness of $\otimes$ we encounter that now there is only one choice for selecting deterministic functions from the $\{b_k\}$, namely the b_k themselves. Following the proof further we get a function $f \in \otimes\{b_K\}$. f is the fixed point of a function g' that was uniquely constructed from the b_k. Banach's fixed point theorem guarantees that this fixed point is unique, hence $\otimes\{b_K\} = \{f\}$. This means that the nested composition of b_K again is deterministic. Furthermore, it is delayed due to the previous theorem. □

Theorem 11 *$\otimes$ is monotonic w.r.t. refinement.*

Proof 11 We have to show that if for all $k \in K$ with $F_k \in B_K$ and $G_k \in D_K$ it holds that $F_k \subseteq G_k$, then $\otimes B_K \subseteq \otimes D_K$. This follows easily from the definition of $\otimes$. □

B.2. Sequential Composition

Theorem 12 *The sequential composition $\otimes B_K$ of the behaviors in B_K is well-defined.*

Proof 12 Using the assumptions in the definition of sequential composition (Section 3.4) we have to prove that there is at least one deterministic behavior $f \in \otimes B_K$.

First we select a deterministic behavior $f_k \in F_k$ for every $k \in K$. Then, we define f as follows: For all $k \in K$ $f(i)|_{O_k} = o_k$, where o_k is the unique element in O_k such that $o_k = f_k((i + \Sigma_{j<k} o_j)|_{I_k})$. Using well-founded induction over K we can prove that for every $k \in K$, such a unique o_k exists. Hence, f is well-defined.

Again with well-founded induction and by definition of f we can show that for given $i{\downarrow}t = i'{\downarrow}t$ it holds that $f(i){\downarrow}t = f(i'){\downarrow}t$. Therefore, f also is a behavior.

From the definition of $\otimes$ and the restrictions on the dependencies between behaviors F_k and F_j for $j, k \in K$, which are imposed by sequential composition, it follows that $f \in \otimes B_K$. $\square$

Theorem 13 *The strict sequential composition* $(;B_K)$ *of the behaviors in* B_K *has a delay* $\delta(;B_K) \geq \Sigma_{k \in K}\delta(F_k)$.

Proof 13 We show that for all $f \in (;B_K)$, $\delta(f) \geq \Sigma_{k \in K}\delta(F_k)$. Let f be in $;B_K$ and let $f_k \in F_k$ be those functions from which f is constructed by the definition of $;B_K$. Furthermore, let $g \in \otimes B_K$ be that function from which f results by interface adaption, i.e. $f = g \updownarrow_{I_0}^{O_n}$. By induction we can prove that if $o = g(i)$ and $o' = g(i')$ then for all $k \in K = \{0, \ldots, n\}$, it holds that

$$o|_{O_k}\downarrow(t + \Sigma_{j=0}^{k}\delta(f_j)) = o'|_{O_k}\downarrow(t + \Sigma_{j=0}^{k}\delta(f_j))$$

if $i|_{I_0}\downarrow t = i'|_{I_0}\downarrow t$. $\square$

Note that in a sequential composition the predecessor of a component need not produce all possible outputs (i.e. it may may not be surjective) and therefore the sequential composition may exhibit more delay than the sum of the delays of its elements.

B.3. Parallel Composition

Theorem 14 *The parallel composition* $\otimes B_K$ *of the behaviors in* B_K *is well-defined.*

Proof 14 Using the assumptions in the definition of parallel composition (Section 3.5) we have to prove that there is at least one deterministic behavior $f \in \otimes B_K$.

For every $k \in K$ let f_k be in F_K. We define f as follows: $f(i) = o$ exactly if for all $k \in K$ $o|_{O_k} = f_k(i|_{I_k})$. f is well defined as all the output channels of the f_k are disjoint. With the assumptions imposed on the F_k by the definition of parallel composition, we immediately get $f \in \otimes B_K$.

From the construction of f we can easily derive that f is a behavior, because all the f_k are behaviors by definition. $\square$

Theorem 15 *The delay of the parallel composition of the behaviors in* B_K *is* $\delta(\otimes B_K) = inf\{\delta(F_k) | k \in K\}$.

Proof 15 First, observe that every $f \in \otimes B_K$ can be constructed from some $f_k \in F_K$ in the same way as in the previous proof. From this construction we easily derive that f has delay δ if every f_k is delayed with δ. Hence, $\delta(f) \geq \mathit{inf}\{\delta(f_k) \mid k \in K\}$. As parallel composition uses each composed function in all its arguments and no hiding occurs, $\mathit{inf}\{\delta(f_k) \mid k \in K\}$ is indeed a sharp bound for the parallel composition f.

Using this result it is easy to derive that the above equality holds. $\square$

B.4. Feedback

Theorem 16 *The feedback $\uparrow F$ of behavior $F \in \wp(\overrightarrow{I_F} \to \overrightarrow{O_F})$ is well-defined if F is delayed on the feedback channels.*

Proof 16 The theorem only demands that F is delayed on the feedback channels $C = I_F \cap O_F$, i.e. that there is a $\delta > 0$ such that for all i, i' with $i|_C{\downarrow}t = i'|_C{\downarrow}t$ and $i|_{I_F \setminus C}{\downarrow}(t+\delta) = i'|_{I_F \setminus C}{\downarrow}(t+\delta)$ it holds that $F(i)|_C{\downarrow}(t+\delta) = F(i')|_C{\downarrow}(t+\delta)$. It need not be delayed on the other channels.

To show that $\uparrow F$ is nonempty we select a deterministic behavior $g \in F$. The rest of the proof now follows the proof of Theorem 8, with the only difference that the function from which the parameterized fixed point f is now constructed is only delayed in its first argument and not in all inputs. Nevertheless, this is all that is needed to ensure the existence of the parameterized fixed point on a complete metric space. Again we get that $\delta(f) \geq \delta(g)$.

By construction $f \in \otimes\{F\}$. Thus, $f \updownarrow_I^O \in \uparrow F$. $f \updownarrow_I^O$ is a behavior with $\delta(f \updownarrow_I^O) \geq \delta(g)$, because interface adaption can only increase the delay. $\square$

Theorem 17 $\delta(\uparrow F) \geq \delta(F)$

Proof 17 is quite similar to previous arguments and therefore left to the reader. $\square$

C. Kind Respecting Behaviors

Theorem 18 *The set of delayed, kind respecting behaviors with at least delay δ for a given $\delta > 0$ is closed with respect to nested composition, parallel composition, sequential composition, feedback, renaming and interface adaption.*

Proof 18 As $\overset{\ominus\!\!\to}{C} \subseteq \overrightarrow{C}$ and all functions in $\overset{\ominus\!\!\to}{I} \overset{\delta}{\to} \overset{\ominus\!\!\to}{O}$ can be extended to functions in $\overrightarrow{I} \overset{\delta}{\to} \overrightarrow{O}$, all the above mentioned operations carry over to kind respecting behaviors. It therefore suffices to prove the closedness of the operations, i.e. we have to prove that the composition of kind respecting behaviors with at least delay $\delta > 0$ is again a kind respecting behavior with at least delay δ.

For renaming and interface adaption, this is clear from the context conditions. As feedback, sequential and parallel composition are special cases of nested composition, it suffices that nested composition is closed.

We have already proven that the result of a nested composition has a delay which is at least the infimum of the delays of its components (see Theorems 8 and 9).

Now let a system of components $B_K^\circ = \{F_k^\circ | k \in K\}$ with delayed kind respecting behaviors $F_k^\circ \in \wp(\overset{\ominus\!\!\to}{I_k} \overset{\delta}{\to} \overset{\ominus\!\!\to}{O_k})$ with $\delta > 0$ be given. These functions can be extended to a system of functions $B_K = \{F_k | k \in K\}$ with delayed behaviors $F_k \in \wp(\overrightarrow{I_k} \overset{\delta}{\to} \overrightarrow{O_k})$ in a pointwise manner: F_k is precisely characterized by $F_k|_{\overset{\ominus\!\!\to}{I_k}} = F_k^\circ$. Please note the equality which essentially says that each function in F_k° can be extended in at least one way.

We can now apply the nested composition operation $\otimes$. By the definition of $\otimes B_K$ we get that each $f \in \otimes B_K$ is constructed from some $f_k \in F_k$ for all $k \in K$. According to the construction of F_k there exist according functions $f_k^\circ \in F_k^\circ$ that are appropriate restrictions, i.e. $f_k^\circ = f_k|_{\overset{\ominus\!\!\to}{I_k}}$.

Defining $f^\circ = f|_{\overset{\ominus\!\!\to}{I}}$ as the restriction of f to inputs of the right kind, we have to prove that f° is kind respecting. In the following we use induction over time t in order to do so.

Let us assume the kind respecting input $i \in \overset{\ominus\!\!\rightarrow}{I}$ be given. The induction assumption is that up to time $t \in \mathbb{R}_+$ (t exclusive) no violation of a channel kind happened on any output channel.

This is certainly true for $t = 0$ as the channel kinds we defined can only be violated in a nonempty time interval, not in an empty one.

As all components f_k are kind respecting and delayed with δ it follows that all channels controlled by any of these components are kind respecting up to time $t + \delta$.

This indeed ensures that no violation happens in finite time. As channel kinds are defined in such a way that their violation can already be detected on a finite interval, this means the resulting composition f is kind respecting which implies that $\otimes B_K$ is kind respecting.

It is also possible to prove that $\otimes B_K$ is kind respecting by going back to the definition of $\otimes$ and using that $\overset{\ominus\!\!\rightarrow}{I}$ and $\overset{\ominus\!\!\rightarrow}{O}$ are complete metric spaces (Section A.2): As in proof 8 we construct functions g and g' and define $f' = \mu g'$ for any function $f \in \otimes B_K$. By the definition of nested composition it is easy to show that $f(i) = f'(i)$ for $i \in \overrightarrow{I}$. We now define g'° as the restriction of g' to $\overset{\ominus\!\!\rightarrow}{O} \times \overset{\ominus\!\!\rightarrow}{I}$. The way g' was constructed from kind preserving functions f_k ensures that g'° is delayed and kind respecting, i.e. $g'^{\circ} \in \overset{\ominus\!\!\rightarrow}{O} \times \overset{\ominus\!\!\rightarrow}{I} \to \overset{\ominus\!\!\rightarrow}{O}$. Hence, it is a delayed function on complete metric spaces which implies that for all $i \in \overset{\ominus\!\!\rightarrow}{I}$ there uniquely exists an $o \in \overset{\ominus\!\!\rightarrow}{O}$ such that $o = \mu g'^{\circ}(i)$. Furthermore, we know that $\mu g'^{\circ}$, $\mu g'$, f and f° coincide for inputs in $\overset{\ominus\!\!\rightarrow}{I}$. Therefore, $f^{\circ}(i) \in \overset{\ominus\!\!\rightarrow}{O}$ for all $i \in \overset{\ominus\!\!\rightarrow}{I}$ which implies that $\otimes B_K$ is kind respecting. $\square$

Note that the set of kind respecting behaviors with delays $\delta \geq 0$ is also closed w.r.t. parallel composition, sequential composition, renaming and interface adaption.

D. Abstraction of Behaviors

Theorem 19 *The set $\mathcal{T}_{\nabla_T}(\overrightarrow{I})$ is non-empty for the ∇_T operators defined in Sections 5.2.1 and 5.2.2.*

Proof 19 For each of the ∇_T operators from Sections 5.2.1 and 5.2.2 we define a construction $c \in \tilde{I} \rightarrow \overrightarrow{I}$ such that $c(\tilde{I})$ is in $\mathcal{T}_{\nabla_T}(\overrightarrow{I})$. The construction c can be regarded as a inverse function of ∇_T that selects one specific $y \in \overrightarrow{I}$ with $\nabla_T(y) = x$.

First we consider the abstraction operator from message streams to timed streams (Section 5.2.1). For every discrete timed stream $x \in \tilde{I}$ we define $c(x) \in \overrightarrow{I}$ such that the restriction of $c(x)$ to the interval $[n \cdot T, (n+1) \cdot T)$ contains exactly the messages in the list $x(n+1)$, in the same order. The time instance at which the m-th message of the list $x(n+1)$ occurs is defined as $t_{n,m} = n \cdot T + \Sigma_{k=1}^{m}(\frac{T}{1+T})^k$. Note that $t_{n,m} < (n+1) \cdot T$ for $m < \infty$, because of the limit value of the given series. With this definition of $c(x)$ is is easy to show that $c(\tilde{I}) \in \mathcal{T}_{\nabla_T}(\overrightarrow{I})$.

Now we consider the abstraction operator from step streams to timed streams (Section 5.2.2). Here, we define $c(x)(t) = x(\lfloor \frac{t}{T} \rfloor)$ if $t \geq T$ and $c(x)(t) = e$ if $t < T$, where e is some element in I. Again it is easy to prove $c(\tilde{I}) \in \mathcal{T}_{\nabla_T}(\overrightarrow{I})$. $\square$

Lemma 1 *The abstraction operators ∇_T of Sections 5.2.1 and 5.2.2 preserve equal prefixes: $x_1 \downarrow t = x_2 \downarrow t \Rightarrow \nabla_T(x_1) \downarrow \lfloor \frac{t}{T} \rfloor = \nabla_T(x_2) \downarrow \lfloor \frac{t}{T} \rfloor$*

Proof 20 The proof is obvious. $\square$

Theorem 20 *If behavior F is delayed with $\delta > 0$ then its discrete time abstraction $G = \nabla_T(F)$ is delayed with $max\{n \mid n \cdot T \leq \delta\}$ for ∇_T as defined in Section 5.2.1 or 5.2.2.*

Proof 21 Let g be an arbitrary element of $\nabla_T(F)$ and $y_1, y_2 \in \tilde{I}$ two timed streams with equal prefixes, $y_1 \downarrow n = y_2 \downarrow n$ for $n \in \mathbb{N}$. By definition g is constructed from some $f \in F$ and $X \in \mathcal{T}_{\nabla_T}(\overrightarrow{I})$. As $\nabla_T \in X \to \tilde{I}$ is surjective, there are $x_i \in X$ with $y_i = \nabla_T(x_i)$, $i \in \{1, 2\}$. Due to the definition of $\mathcal{T}_{\nabla_T}(\overrightarrow{I})$ it follows that $x_1{\downarrow}(nT) = x_2{\downarrow}(nT)$. Thus, $f(x_1){\downarrow}(nT+\delta) = f(x_2){\downarrow}(nT+\delta)$ holds, which implies $f(x_1){\downarrow}(nT+mT) = f(x_2){\downarrow}(nT+mT)$ for $m = max\{k \mid k \cdot T \leq \delta\}$. Applying the abstraction operator and the previous lemma yields $g(y_1){\downarrow}(n+m) = g(y_2){\downarrow}(n+m)$. $\square$

Corollary 3 *The (discrete) time abstraction of a (non-empty) behavior* $F \in \wp(\overrightarrow{I} \xrightarrow{\delta} \overrightarrow{O})$ *is non-empty.*

Proof 22 The corollary is a consequence of the preceding theorems. $\square$

Bibliography

[BDD+92] Manfred Broy, Frank Dederich, Claus Dendorfer, Max Fuchs, Thomas Gritzner, and Rainer Weber. The design of distributed systems - an introduction to FOCUS. Technical Report TUM-I9202, Technische Universität München, 1992.

[BDD+93] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The Design of Distributed Systems: An Introduction to Focus — Revised Version. Technical Report TUM-I9202-2, Technische Universität München, 1993.

[Ber98] G. Berry. The Foundations of Esterel. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1998.

[BFG+93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hußmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Specification Language Spectrum, An Informal Introduction, Version 1.0, Part 1. Technical Report TUM-I9312, Technische Universität München, 1993.

[BHH+97] Ruth Breu, Ursula Hinkel, Christoph Hofmann, Cornel Klein, Barbara Paech, Bernhard Rumpe, and Veronika Thurner. Towards a formalization of the unified modeling language. In Satoshi Matsuoka Mehmet Aksit, editor, *ECOOP'97 Proceedings*. Springer-Verlag, LNCS 1241, 1997.

[Bro93] M. Broy. Interaction Refinement – The Easy Way. In M. Broy, editor, *Program Design Calculi*, volume 118 of *NATO ASI Series F: Computer and System Sciences*. Springer-Verlag, 1993.

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

[Eng77] R. Engelking. *General Topology*. PWN - Polish Scientific Publishers, 1977.

[FELR97] Robert France, Andy Evans, Kevin Lano, and Bernhard Rumpe. Developing the UML as a formal modelling notation. In Jean Bezivin and Pierre-Allain Muller, editors, *UML'97 Proceedings*. Springer-Verlag, LNCS, 1997.

[FELR98] Robert France, Andy Evans, Kevin Lano, and Bernhard Rumpe. The UML as a formal modeling notation. *Computer Standards & Interfaces*, to appear, 1998.

[GKR96] Radu Grosu, Cornel Klein, and Bernhard Rumpe. Enhancing the SysLab system model with state. Technical Report TUM-I9631, Technische Universität München, 1996.

[GR95] Radu Grosu and Bernhard Rumpe. Concurrent timed port automata. Technical Report TUM-I9533, Technische Universität München, 1995.

[Gro97a] The UML Group. UML Metamodel. Version 1.1, Rational Software Corporation, Santa Clara, CA-95051, USA, September 1997.

[Gro97b] The UML Group. Unified Modeling Language. Version 1.1, Rational Software Corporation, Santa Clara, CA-95051, USA, July 1997.

[GSB98] 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)*, volume 1486 of *LNCS*. Springer-Verlag, 1998.

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[Hog89] D. Hogrefe. *Estelle, LOTOS und SDL: Standard–Spezifikationssprachen für verteilte Systeme*. Springer-Verlag, 1989.

[IT96] ITU-T. *Z.120 – Message Sequence Chart (MSC)*. ITU-T, Geneva, 1996.

[KRB96] C. Klein, B. Rumpe, and M. Broy. A stream-based mathematical model for distributed information processing systems - SysLab system model - . In Jean-Bernard Stefani Elie Naijm, editor, *FMOODS'96 Formal Methods for Open Object-based Distributed Systems*, pages 323–338. ENST France Telecom, 1996.

[MPS86] D. MacQueen, G. Plotkin, and R. Sethi. An Ideal Model for Recursive Polymorphic Types. *Information and Control*, 71:95–130, 1986.

[MS97] Olaf Müller and Peter Scholz. Functional specification of real-time and hybrid systems. In *Proc. Hybrid and Real-Time Systems, Grenoble, March 26-28, 1997*, volume 1201 of *LNCS*. Springer-Verlag, 1997.

[MT75] M.D. Mesarovic and Y. Takahara. *General Systems Theory: Mathematical Foundations*, volume 113. Academic Press, 1975. Mathematics in Science and Engineering.

[PR97] J. Philipps and B. Rumpe. Refinement of information flow architectures. In M. Hinchey, editor, *ICFEM'97*. IEEE CS Press, 1997.

[Rei90] W. Reisig. *Petrinetze, Eine Einführung*. Springer-Verlag, 1990.

[RKB95] B. Rumpe, C. Klein, and M. Broy. Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme: SysLab Systemmodell. Technical Report TUM-I9510, Technische Universität München, March 1995.

[Rum96] Bernhard Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme*. Ph.D. thesis (in German), Technische Universität München, 1996.

[Rum98] Bernhard Rumpe. A note on semantics (with an emphasis on UML). In Haim Kilov and Bernhard Rumpe, editors, *Second ECOOP Workshop on Precise Behavioral Semantics*. Technische Universität München, TUM-I9813, 1998.

[Sch98] P. Scholz. *Design of Reactive Systems and their Distributed Implementation with Statecharts*. Ph.D. thesis, Technische Universität München, 1998.

[SG90] D. Scott and C. Gunter. Semantic Domains and Denotational Semantics. In *Handbook of Theoretical Computer Science*, chapter 12, pages 633 – 674. Elsevier Science Publishers, 1990.

[Sut75] W. A. Sutherland. *Introduction to metric and topological spaces*. Claredon Press - Oxford, 1975.

[vdB94] M. von der Beeck. A comparison of statecharts variants. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *LNCS*, pages 128 – 148. Springer-Verlag, 1994.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, 1993.

[Wir90] M. Wirsing. Algebraic Specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 675–788. Elsevier Science Publishers, 1990.