Samarjit Chakraborty
Jörg Eberspächer *Editors*

Advances in Real-Time Systems

Springer

Advances in Real-Time Systems

Samarjit Chakraborty • Jörg Eberspächer
Editors

Advances in Real-Time Systems

Springer

Editors
Samarjit Chakraborty
TU München
LS für RealzeitComputersysteme
Arcisstr. 21
80290 München
Germany
Samarjit.Chakraborty@rcs.ei.tum.de

Prof. Dr. Jörg Eberspächer
TU München
LS Kommunikationsnetze
Arcisstr. 21
80290 München
Germany
joerg.eberspaecher@tum.de

ISBN 978-3-642-24348-6 e-ISBN 978-3-642-24349-3
DOI 10.1007/978-3-642-24349-3
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011942997

Mathematics Subject Classification (2000): 01-01, 04-01, 11Axx, 26-01

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Georg Färber
on the occasion of his appointment as
Professor Emeritus at TU München after 34
illustrious years as the Head of the Lehrstuhl
für Realzeit-Computersysteme

Preface

This book is a tribute to Georg Färber on the occasion of his appointment as Professor Emeritus at TU München after 34 years of service as the Head of the Lehrstuhl für Realzeit-Computersysteme.

Georg Färber was born in 1940 and obtained his PhD in 1967 from TU München. In an illustrious career, spanning over 40 years, Prof. Färber contributed significantly to the area of real-time and embedded systems, in terms of both education and research. His early research dealt with specification and design of embedded systems and mapping of such specifications onto appropriate hardware and software architectures. In particular, he was active in the area of computer-aided process control, with a focus on distributed and fault-tolerant systems. Later, his interests broadened into real-time capturing and interpretation of visual information, especially in the context of robot vision. More recently, he has worked in the area of cognitive automobiles, e.g., driver assistance systems and autonomous cars. Since a number of years, Prof. Färber also took a keen interest in topics at the intersection of engineering and medicine, in particular in those related to "e-health".

In parallel to his research and teaching activities, Prof. Färber was a highly successful entrepreneur. In 1969, along with his brother Eberhard Färber, he founded PCS-Computersysteme GmbH. This company was once considered to be among the most innovative IT companies in Munich and led to more than 20 other spin-offs. Among other products, PCS developed the first UNIX Workstations in Germany (named CADMUS), which were the only European alternatives to US-based products for a long time. In 1986, Mannesmann/Kienzle became the majority stakeholder of PCS, and Georg Färber provided the technical leadership during 1988-89, while on leave from TU München.

In addition to authoring one of the earliest books on real-time systems, Georg Färber served as the editor of the journal "Information Technology" and is a member of the Board of Trustees of the Fraunhofer-Institute for Information and Data Processing (Fraunhofer IITB). He has also served in various – often advisory – capacities at the DFG, the Max Planck Society, and in several other scientific and industrial councils and German government agencies.

Given Georg Färber's remarkable achievements and his reputation, we invited a number of well-known researchers to contribute a collection of chapters reflecting the state of the art in the area of real-time systems. These chapters cover a variety of topics spanning over automotive software and electronics, software timing analysis, models for real-time systems, compilation of real-time programs, real-time microkernels, and cyber-physical systems. We believe that this collection can serve as a reference book for graduate-level courses. It will also be helpful to both researchers in the academia and practitioners from the industry.

Munich, Germany

Samarjit Chakraborty
Jörg Eberspächer

Contents

Part I
Theoretical Foundations

Chapter 1
System Behaviour Models with Discrete and Dense Time

Manfred Broy

1.1 Introduction and Motivation

The notion of *system* is present in many scientific disciplines. Biology speaks of *biological system*. There are terms like *economic system*, *ecological system*, *logical system*. The whole world can be understood as a *dynamical system*. Describing systems, their structure and their dynamics by appropriate models is a major goal of scientific disciplines. However, the different disciplines use quite different notions, concepts, and models of systems. Mathematics, has developed differential and integral theory over the centuries as one way of modelling and studying systems in terms of mathematical models. Relevant system aspects are captured by real valued variables that change dynamically and continuously depending on the parameter of time. This way the system dynamics and the dependencies between the system variables can be described by differential and integral equations. The engineering discipline of modelling and designing systems applying these mathematical modelling concepts is control theory.

Another way to capture and specify systems in terms of discrete events is logic. Logic was developed originally as a branch of philosophy addressing the art and science of reasoning. As a discipline, logic dates back to Aristotle, who established its fundamental place in philosophy. Historically, logic was intended as a discipline of capturing ways of thinking of human beings aiming at crisp lines of arguments. Over the centuries, logic was further developed mainly as a basis for carrying out proofs by formal logical deduction, leading to mathematical logic with its foundations *propositional logic* and *predicate logic* in its many variations. With the arrival of digital systems, logic became more and more also a technical discipline

M. Broy (✉)
Institut für Informatik, Technische Universität München, 80290 München, Germany
e-mail: broy@in.tum.de

S. Chakraborty and J. Eberspächer (eds.), *Advances in Real-Time Systems*,
DOI 10.1007/978-3-642-24349-3_1,

for designing logical circuits and electronic devices. With software becoming more significant in all kinds of information processing applications, more general forms of logic were invented as the conceptual basis for engineering information processing systems. For information processing systems, including embedded systems, many different aspects are captured by various forms of logic both at the technical level and the application domain level.

The logic of the behaviour of complex discrete event systems can be captured by families of discrete events that are causally related with logically specified properties.

In contrast to real-valued function based models of systems such as applied in control theory, by logics we can capture better ways of arguing about systems. When capturing requirements about systems such as in requirements engineering in terms of natural language, a logical way of making requirements precise is more appropriate than modelling behaviours by real time parameterized continuous functions. On the other hand, when solving problems in control theory we finally aim at mathematical descriptions of the system dynamics by differential and integral equations.

In this paper we aim at a step integrating logical system views with system views based on differential and integration calculus such as used in control theory. In contrast to well-known approaches, where the step from a description of systems by real valued functions into digital systems is performed using techniques of discretization such as in numerical analysis, we are rather interested in the step from a logical description of requirements to modelling of system behaviours by continuous real valued functions and in a formal relationship between the logical description of requirements and the real valued functions describing systems dynamics.

We are aiming at systems that interact with their environment. Streams of data for exchanging input and output capture this interaction. We consider both discrete and continuous streams.

One way to describe interactive systems is state machines with input and output also known as Mealy or Moore machines. These are machines, where in given states input triggers transitions generating new states and output. I/O state machines define computations, being infinite runs by their transitions. Given an initial state and an infinite stream of input messages I/O machines produce infinite streams of states and infinite streams of output messages. In so-called interface abstractions we forget about the chain of states of computations and just keep the relation between the input and output streams. This yields what we call an *interface abstraction*. Talking only about interface properties we can formulate so-called *interface assertions*, which describe logical relationships between the input and the output streams. They specify the interface behaviour of I/O state machines.

In the following, we aim at techniques for modelling interfaces of discrete as well as dense and continuous interactive systems. We define models of such systems and discuss concepts how to specify, compose and relate them.

1.2 Hybrid Interactive Behaviours

The essential difference between a *non-interactive* and an *interactive computation* lies in the way in which input is provided to the computing device before or during the computation and how output is provided by the computing device to its environment during or after the computation.

1.2.1 Streams, Channels, and Histories

In this section we briefly introduce the notions of stream, channel, and history.

Throughout this paper, we model interaction by message exchange over sequential communication media called *channels*. These message streams can be based on discrete or dense time and may be discrete or – in the case of dense time – continuous. In general, in an interactive computation, several communication channels may participate. In our setting, a channel is simply an identifier for a communication line. In the following, we distinguish input from output channels. Throughout the paper, let I be a set of input channels, O be a set of output channels and $\mathtt{M}$ be a set of messages.

A stream may be discrete or continuous. A stream has a data type that determines the type of messages it carries. A stream can be finite or infinite.

1.2.1.1 Discrete Finite and Infinite Streams

Let $\mathtt{M}$ be a set of elements, called *messages*. We use the following notation (where set $\mathbb{N}_+$ is specified by $\mathbb{N}_+ = \mathbb{N}\backslash\{0\}$):

$\mathtt{M}^*$ denotes the set of finite sequences, with elements of the set $\mathtt{M}$, including the *empty* sequence $\langle\rangle$,
$\mathtt{M}^\infty$ denotes the set of infinite sequences with elements of the set $\mathtt{M}$ (that are represented by the total mappings $\mathbb{N}_+ \to \mathtt{M}$).

By

$$\mathtt{M}^\omega = \mathtt{M}^* \cup \mathtt{M}^\infty$$

we denote the set of discrete untimed streams of elements of set $\mathtt{M}$. Streams of elements of set $\mathtt{M}$ are finite or infinite sequences of elements of set $\mathtt{M}$.

The set of streams has a rich algebraic and topological structure. We make use of only parts of this structure. We introduce concatenation $\hat{}$ as an operator on streams written in infix notation:

$$\_\hat{}\_ : \mathtt{M}^\omega \times \mathtt{M}^\omega \to \mathtt{M}^\omega$$

On finite streams concatenation is defined as usual on finite sequences. For infinite streams r, s: $\mathbb{N}_+ \to \mathtt{M}$ and finite stream $x \in \mathtt{M}^*$ we define the result of concatenation for infinite streams as follows:

$$s\hat{}x = s$$

$$s\hat{}r = s$$

$$\langle x_1 \ldots x_n\rangle\hat{}\langle s_1 \ldots\rangle = \langle x_1 \ldots x_n s_1 \ldots\rangle$$

We may represent finite streams by total functions $\{1, \ldots, t\} \to M$ or also by partial functions $\mathbb{N}_+ \to M$ and infinite streams by total functions $\mathbb{N}_+ \to M$.

Streams are used to represent the flow of messages sent over a communication channel during the lifetime of a system. Of course, in concrete physical systems this communication takes place in a specific time frame. Hence, it is often convenient or even essential to be able to refer to time. Moreover, with an explicit notion of time the theory of feedback loops in networks of communicating components gets even simpler (see [1]). Therefore we prefer to work with *timed streams*.

Streams represent histories of communications of data messages transmitted within a time frame. Given a message set M of type T a *uniformly discretely timed stream* is a function

$$s : \mathbb{N}_+ \to M^* \quad \text{i.e. } s \in (M^*)^\infty$$

Actually, given stream $s \in (M^*)^\infty$ for every time interval $t \in \mathbb{N}_+$ the sequence s(t) denotes the sequence of messages communicated in the time interval t in the stream s. Let $\mathbb{R}_+ = \{t \in \mathbb{R} : t \geq 0\}$ be the set of positive real numbers. The basic idea here is that s(t) represents the sequence of messages communicated in the real time interval $[(t-1)\delta : t\delta[$ where $\delta \in \mathbb{R}_+$ is called the *time granularity* of the stream s. A *partial stream* is given for $t \in \mathbb{N}$ by a mapping

$$s : \{1, \ldots, t\} \to M^* \quad \text{i.e. } s \in (M^*)^*$$

It represent a communication history till time step t. Throughout this paper we work with a number of simple basic operators and notations for streams and timed streams respectively that are briefly summarized below:

$\langle\rangle$	empty sequence or empty stream,
$\langle m\rangle$	one-element sequence containing m as its only element,
s.t	t-th element of the stream s (which is a message in case s is an untimed stream and a finite sequence of messages in case s is a timed stream),
#s	length of a stream
$s{\downarrow}t$	prefix of length t of the stream s (which is a sequence of messages of length t, provided $\#s \geq t$, in the case of an untimed stream and a sequence of t sequences in case s is a discretely timed stream),
$s \uparrow t$	the stream derived from s by deleting its first t elements (without the first t sequences of s in the case of a discretely timed streams)

In a uniformly discretely timed stream $s \in (M^*)^\infty$ it is specified in which time intervals which sequences of messages are transmitted. The timing of the messages within a time interval is not specified, however, only their order is observable.

1.2.1.2 Dense and Discrete Time Domains and Timed Streams

Every subset TD $\subseteq \mathbb{R}_+$ is called a *time domain*. A time domain TD is called *discrete,* if for every number $t \in \mathbb{R}_+$ the set

$$TD_t = \{x \in TD : x < t\}$$

is finite. Obviously, discrete time domains contain minimal elements.

A set S with a linear order $\leq$ is called *dense*, if the following formula holds

$$\forall x,\ y \in S : x < y \Rightarrow \exists z \in S : x < z < y$$

On $\mathbb{R}$ we choose the classical linear order $\leq$. If a nontrivial time domain TD is discrete, it is certainly not dense. Vice versa, however, if a time domain is not dense, it is not necessarily discrete.

Let M be a set of messages and TD be a time domain. A timed stream over time domain TD is a total mapping

$$s : TD \rightarrow M$$

TD is called the (time) domain of the stream s. We write dom(s) = TD. If TD is discrete, then the stream s is called *discrete*, too. If TD is dense, then s is called *dense*, too.

Stream s is called *continuous*, if TD is an interval in $\mathbb{R}_+$ and M is a metric space with distance function d such that s is a continuous function on the set TD in Cauchy's sense. More precisely s is continuous in $t \in \mathbb{R}_+$ if the following formula holds:

$$\forall \varepsilon \in \mathbb{R}, \varepsilon > 0 : \exists \delta \in \mathbb{R}, \delta > 0 : \forall x' \in TD : |x - x'| < \delta \Rightarrow d(s(x), s(x')) < \varepsilon$$

An interval based timed stream s is given by an interval [t: t'[with $t, t' \in \mathbb{R}_+$, $t \leq t'$, and by the time domain TD $\subseteq$ [t: t'[where

$$s : [t : t'[\rightarrow M$$

is a partial function and TD is its domain. We write then interval(s) = [t: t'[and dom(s) = TD. Note that for the partial functions we denote by interval(s) the set of potential arguments for function s while dom(s) $\subseteq$ [t: t'[defines the set of arguments for which function s is defined.

A stream s is called *permanent*, if interval(s) = dom(s). Then s is a total function on its interval(s).

A special case of a permanent stream s is one who is piecewise *constant* with only a discrete set of "discontinuities". Then we expect a discrete set of time points (with $t_0 = 0$):

$$\{t_i : i \in \mathbb{N}\}$$

such that in each interval $[t_i : t_{i+1}[$ the stream s is constant, i.e. $s(t) = d$ with some $d \in T$ where T is the type of the stream s. A typical example is $T = \mathbb{B}$ where stream s(t) signals whether a certain condition holds or does not at time t.

1.2.1.3 Operations on Timed Streams

For timed streams we define a couple of simple basic operators and notations that are summarized below:

$\langle\rangle$	empty sequence or empty stream with $dom(TD) = \emptyset$ and $interval(\langle\rangle) = [0 : 0[= \emptyset$,
$\langle m@t\rangle$	one-message stream containing m as its only message at time t with $dom(\langle m@t\rangle) = \{t\}$,
#s	number of messages in a stream (which is given by the cardinality $\lvert dom(s)\rvert$)
s.j	j-th element in the discrete stream s (which is uniquely determined provided $\#s \geq j$ holds),
$s{\downarrow}t$	prefix until time t of the stream s (which is also denoted by $s\lvert dom(s) \cap [0: t[$ where by $f\lvert M$ denotes the restriction of a function $f: \mathtt{D} \to \mathtt{R}$ to the set $M \subseteq \mathtt{D}$ of arguments),
$s{\uparrow}t$	the stream derived from stream s by deleting its messages until time t (which is $s\lvert dom(s) \setminus (dom(s) \cap [0: t[)$) with domain $dom(s) \setminus (dom(s) \cap [0: t[$ and the interval $interval(s) \setminus interval(s{\downarrow}t)$

Let $interval(s) = [0: t[$; by $s{\downarrow}t'$ we get a stream for the interval $[0: t'[$ and with $dom(s{\downarrow}t') = dom(s) \cap [0: t'[$. By PTS we denote the set of partial timed streams. Given time $t \in \mathbb{R}_+$, we denote by PTS(t) the set of partial streams that are either discrete with $dom(s) \subseteq [0: t[$ or that are permanent with domain $dom(s) = [0: t[$.

Given times $t, t' \in \mathbb{R}_+$ by $PTS[t: t'[$ we denote the set of partial streams with $interval(s) = [t: t'[$ that are either discrete with $dom(s) \subseteq [t: t'[$ or that are permanent on their domain $dom(s) = [t: t'[$.

This way we get the universe of streams over a given universe of messages types.

A *time shift* of a timed stream s by the time $u \in \mathbb{R}_+$ yields stream $s^{TM}u$ defined by the equations

$$interval(s^{TM}u) = [t + u : t' + u[\Leftarrow interval(s) = [t : t'[$$
$$dom(s^{TM}u) = \{t + u : t \in dom(s)\}$$

and for $t \in dom(s)$ defined by the equation

$$(s^{TM}u)(t + u) = s(t)$$

Given a stream s with interval $[t: t'[$ where $t' < \infty$ (otherwise $s\hat{}s' = s$) we define *concatenation* of stream s with a stream s' by the equation

$$\text{interval}(s\hat{}s') = [t : t''[\Leftarrow \text{interval}(s'^{TM}t') = [t''', t''[\wedge \text{interval}(s) = [t : t'[$$
$$\text{dom}(s\hat{}s') = \text{dom}(s) \cup \text{dom}(s'^{TM}t')$$

and for $t'' \in \text{dom}(s\hat{}s')$

$$(s\hat{}s')(t'') = s(t'') \Leftarrow t'' \in \text{dom}(s)$$
$$(s\hat{}s')(t'') = (s'^{TM}t')(t'') \Leftarrow t'' \in \text{dom}(s'^{TM}t')$$

This generalizes the operations on discrete streams to operations on timed streams.

1.2.1.4 Time Deltas and Delta Transactions

For timed streams their time granularity is of major interest. A discrete stream s has a guaranteed message distance δ if in each time interval of length δ at most one message or event occurs. Formally

$$\forall t \in \mathbb{R}_+ : \#(s|[t : t + \delta[) \leq 1$$

Here s|[t: t + δ[denotes the stream which is the result of restricting stream s (seen as a mapping) to the set [t: t + δ[.

In a time interval of length δ a communication stream is given by a finite sequence of messages, by a continuous function, by a discrete real time sequence, or a mixture thereof.

1.2.2 Channels

Generally, several communication streams may appear in a system. To distinguish and identify these streams we use channels. A channel is a named sequential communication medium. In logical formulas about systems, a channel is simply an identifier in a system that evaluates to a stream in every execution of the system.

Definition 1.1. Channel snapshot and channel history

Let C be a set of channels; given times t, t' $\in \mathbb{R}_+ \cup \{\infty\}$ with t < t' a channel snapshot is a mapping

$$x : C \to \text{PTS}[t : t'[$$

such that x(c) $\in$ PTS[t: t'[is a partial or total stream for each channel c $\in$ C. A snapshot is called *finite* if t' $< \infty$. By $\widehat{C}$[t: t'[the set of finite channel snapshots for channel set C for times t, t' is denoted. By $\widehat{C}$ the set of all channel snapshots for channel set C is denoted.

A complete channel history is a mapping

$$x : C \to PTS[0 : \infty[$$

such that x(c) is a timed stream for each channel $c \in C$. $\vec{\vec{C}}$ the set of complete channel histories for channel set C. $\vec{C}$ denotes the set of channel histories where all channels carry uniformly discretely timed streams $s \in (M^*)^\infty$. □

All operations and notations introduced for streams generalize in a straightforward way to channel histories applying them to the streams in the histories elementwise.

For instance given a channel history

$$x : C \to PTS[0 : \infty[$$

by x|[t: t‘[we denote a snapshot

$$x|[t : t‘[: C \to PTS[t : t‘[$$

where each stream x(c) for channels $c \in C$ is restricted to the interval [t: t‘[as follows:

$$(x|[t : t‘[)(c) = x(c)|[t : t‘[$$

The remaining operators generalize in analogy from streams to channel histories.

1.2.3 I/O-Behaviours: Interface Behaviours of Hybrid Systems

Let I be a set of typed input channels and O be a set of typed output channels. Figure 1.1 gives an illustration of the system where the channels are represented by arrows annotated with their names and their message types. In addition to the message types that show which elements are communicated via the channels we indicate which type of stream is associated with the channel – a discrete or a dense one. We use the prefix *Dsc* to indicate that a stream is discrete and *Prm* to indicate that it is permanent. For instance *Dsc* Bool is the type of a discrete stream of Boolean values while *Prm* Bool is the type of a permanent stream of Boolean values. If no prefix is used than nothing specific is assumed about the stream.

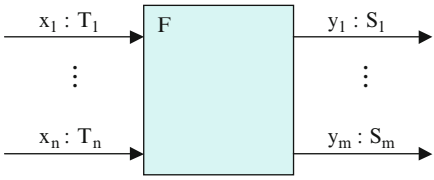

Fig. 1.1 Hybrid system with its channels

We represent hybrid system behaviours by functions:

$$F : \overrightarrow{I} \to \wp(\overrightarrow{O})$$

that model input and output of interactive nondeterministic computations. F maps every input history onto its set of output histories.

Definition 1.2. Causality

An I/O-behaviour F fulfils the property of *causality* if there exists some time distance $\delta \in \mathbb{R}$ with $\delta \geq 0$ such that the following formula holds for all histories $x, z \in \overrightarrow{I}$, $y \in \overrightarrow{O}$, $t \in \mathbb{R}_+$:

$$x{\downarrow}t = z{\downarrow}t \Rightarrow \{y{\downarrow}t + \delta : y \in F(x)\} = \{y{\downarrow}t + \delta : y \in F(z)\}$$

If the formula holds for $\delta = 0$ then F is called *causal* and if it holds for some delay $\delta > 0$ then F is called *strongly causal* and also *strongly δ causal.* □

We assume for I/O-behaviours that they fulfil the law of *strong δ causality* for some $\delta > 0$. Strong causality characterizes proper time flow and the fact that computation takes time. It captures the principle that a reaction to input can happen only after the input has been received. Since the output at time t is produced while the input in step t is provided, the output in step t must depend at most on input provided before time t.

A behaviour F is called *deterministic* if F(x) is a one element set for each input history x. Such a behaviour is equivalent to a function

$$f : \overrightarrow{I} \to \overrightarrow{O} \quad \text{where } F(x) = \{f(x)\}$$

f represents a deterministic I/O-behaviour, provided that for some $\delta \geq 0$ the δ causality property holds. Then the following property is valid:

$$x{\downarrow}t = z{\downarrow}t \Rightarrow f(x){\downarrow}t + \delta = f(z){\downarrow}t + \delta$$

As for nondeterministic behaviours this causality property models proper time flow.

Definition 1.3. Realizability

An I/O-behaviour F is called (strongly) *realizable*, if there exists a (strongly) causal total function

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

such that we have:

$$\forall x \in \overrightarrow{I} : f(x) \in F(x).$$

f is called a *realization* of F. By [F] we denote the set of all realizations of F. An output history $y \in F(x)$ is called *realizable* for an I/O-behaviour F with input x, if there exists a realization $f \in [F]$ with $y = f(x)$. □

A δ causal function $f: \overset{\Rightarrow}{I} \to \overset{\Rightarrow}{O}$ with $\delta > 0$ provides a deterministic *strategy* to calculate for every input history x a particular output history $y = f(x)$. The strategy is called *correct for input x and output y* with respect to an I/O-behaviour F if $y = f(x) \in F(x)$. According to strong δ causality the output y can be computed inductively in an interactive computation. Only input $x{\downarrow}t$ received till time t determines the output till time $t+\delta$ and, in particular, the output at time $t+\delta$. In fact, f essentially defines a deterministic "abstract" automaton with input and output which is, in case of strong δ causality, actually a Moore machine. Strong δ causality guarantees that for each time t the output produced after time t till time $t+\delta$ does only depend on input received before time t.

Theorem 1.1. *Full Realizability*

Strongly δ causal functions $f: \overset{\Rightarrow}{C} \to \overset{\Rightarrow}{C}$ always have unique fixpoints $y = f(y)$.

Proof. This is easily proved by an inductive construction of the fixpoint $y = f(y)$ as follows. Since f is strongly causal output $f(x){\downarrow}\delta$ does not depend on x at all. So we define

$$y{\downarrow}\delta = f(x){\downarrow}\delta$$

for arbitrarily chosen input $x \in \overset{\Rightarrow}{C}$. Then history y is constructed inductively as follows: given

$$y{\downarrow}i\delta$$

we define

$$y{\downarrow}(i+1)\delta = f(x){\downarrow}(i+1)\delta$$

with arbitrary chosen input history x such that

$$x{\downarrow}i\delta = y{\downarrow}i\delta$$

Note again that then $f(x){\downarrow}(i+1)\delta$ does not depend on the choice of the history $x{\downarrow}(i+1)\delta$ due to strong δ causality of f. The construction yields history y such that the fixpoint equation $y = f(y)$ holds. Moreover, since the construction yields a unique result the fixpoint is unique. □

The construction indicates the existence of a computation strategy for the fixpoint y of f.

Definition 1.4. Full Realizability

An I/O-behaviour F is called *fully realizable*, if it is realizable and if for all input histories $x \in \overset{\Rightarrow}{I}$

$$F(x) = \{f(x) : f \in [F]\}$$

holds. Then also every output is realizable. □

Full realizability of a behaviour F guarantees that for all output histories $y \in F(x)$ for some input x there is a strategy that computes this output history. In other words, for each input history x each output history $y \in F(x)$ is realizable.

1.2.4 Hybrid State Machines with Input and Output

In this section we introduce the concept of a hybrid state machine with input and output via channels.

A hybrid state machine (Δ, Λ) with input and output communicated over a set I of input channels and a set O of output channels is given by a state space Σ, which represents a set of states, a set $\Lambda \subseteq \Sigma$ of initial states as well as a state transition function

$$\Delta : (\Sigma \times \widehat{I}) \to \wp(\Sigma \times \widehat{O})$$

For each state $\sigma \in \Sigma$ and each valuation $\alpha \in \widehat{\mathrm{I}}$ of the input channels in I by sequences a snapshot we obtain by every pair $(\sigma', \beta) \in \Delta(\sigma, \alpha)$ a successor state σ' and a valuation $\beta \in \widehat{\mathrm{O}}$ of the output channels consisting of the snapshot of messages produced on the output channels by the state transition. Such state machines are a generalization of *Mealy machines* (more precisely Mealy machines generalized to infinite state spaces and infinite input/output alphabets).

A state machine (Δ, Λ) is called:

- *Deterministic*, if, for all states $\sigma \in \Sigma$ and inputs α, both $\Delta(\sigma, \alpha)$ and Λ are sets with at most one element.
- *Total*, if for all states $\sigma \in \Sigma$ and all inputs α the sets $\Delta(\sigma, \alpha)$ and Λ are not empty; otherwise the machine (Δ, Λ) is called *partial*,
- A (generalized) *Moore machine*, if the output of Δ always depends only on the state and not on the current input of the machine. A Mealy machine is a Moore machine iff the following equation holds for all input sequences α, α' and output sequences β, and all states σ:

$$(\exists \sigma' \in \Sigma : (\sigma', \beta) \in \Delta(\sigma, \alpha)) \Leftrightarrow (\exists \sigma' \in \Sigma : (\sigma', \beta) \in \Delta(\sigma, \alpha'))$$

- *Time based* if the states $\sigma \in \Sigma$ in the state space contain a time attribute denoted by $\mathrm{time}(\sigma) \in \mathbb{R}$ such that for all $(\sigma', \beta) \in \Delta(\sigma, \alpha)$ we have $\mathrm{time}(\sigma) < \mathrm{time}(\sigma')$
- A δ *step timed state machine* for $\delta \in \mathbb{R}$ with $\delta > 0$ if (Δ, Λ) is a time based machine and if for all $(\sigma', \beta) \in \Delta(\sigma, \alpha)$ where

$$\mathrm{time}(\sigma) = \mathrm{j}\delta \text{ and } \mathrm{interval}(\alpha) = [\mathrm{j}\delta{:}(\mathrm{j}{+}1)\delta[$$

we get

$$\mathrm{time}(\sigma') = \mathrm{time}(\sigma) + \delta \text{ and } \mathrm{interval}(\beta) = [\mathrm{j}\delta{:}(\mathrm{j}{+}1)\delta[.$$

Hybrid state machines are a straightforward generalisation of Mealy machines.

1.2.5 Computations of State Machines

In this section we introduce the idea of computations for δ step timed state machines with input and output.

Figure 1.2 shows a computation of a δ step timed state machine with input and output. Actually a computation comprises three infinite streams:

- The infinite streams x of inputs: $x_1, x_2, \ldots \in \widehat{I}$
- The infinite streams y of outputs: $y_1, y_2, \ldots \in \widehat{O}$
- The infinite streams s of states: $\sigma_0, \sigma_1, \ldots \in \Sigma$

Note that every computation can be inductively generated given the input stream x_1, $x_2, x_3, \ldots$ and the initial state $\sigma_0 \in \Lambda$ by choosing step by step state σ_{i+1} and output y_{i+1} by the formula

$$(\sigma_{i+1}, y_{i+1}) \in \Delta(\sigma_i, x_{i+1}).$$

If the state machine is deterministic, then the computation is fully determined by the initial state σ_0 and the input stream x.

Each input history $x \in \overset{\Rightarrow}{I}$ specifies a stream $x_1, x_2, \ldots \in \widehat{I}$ of input snapshots by (for all $j \in \mathbb{N}$)

$$x_{j+1} = x|[j\delta : (j+1)\delta[$$

Given history x a computation of a state machine (Δ, Λ) generates a sequence of states

$$\{\sigma_j : j \in \mathbb{N}\}$$

and a stream $y_1, y_2, \ldots \in \widehat{O}$ of output snapshots where for all times $j \in \mathbb{N}$ we have:

$$(\sigma_{j+1}, y_{j+1}) \in \Delta(\sigma_j, x_{j+1}) \quad \text{and} \quad \sigma_0 \in \Lambda$$

This way every computation specifies an output history $y \in \overset{\Rightarrow}{O}$ that is uniquely specified by

$$y|[j\delta : (j+1)\delta[= y_{j+1}$$

The history y is then called an *output* of the computation of the state machine (Δ, Λ) for input x and initial state σ_0. We also say that the machine computes the output history y for the input history x and the initial state σ_0. This way we can associate an interface behaviour

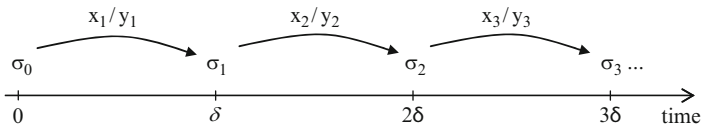

Fig. 1.2 Computation of a δ step timed I/O-machine

$$F_{(\Delta,\Lambda)} : \vec{\vec{I}} \rightarrow \wp(\vec{\vec{O}})$$

with state machine (Δ, Λ) defining $F_{(\Delta,\Lambda)}(x)$ as the set of all histories that are outputs of computations of machine (Δ, Λ) for input history x. System behaviour $F_{(\Delta,\Lambda)}$ is called the *interface abstraction* of hybrid state machine (Δ, Λ).

1.3 Logical Properties of the Interface Behaviour of Hybrid Systems and State Machines

Traditionally temporal logic is used to formulate properties about state transition systems that are represented by state machines. Usually in temporal logic state machines without input and output are considered such that the formulas of temporal logic specify properties for the infinite streams of states generated as computations by these state machines. There are several variations of temporal logic including so-called *linear time temporal logic*, which talks about the state traces of a state machine and *branching-time temporal logic*, which considers trees of computations defined by a state machine.

Since we are not mainly interested in states but rather in interface behaviour in terms of input and output streams of computations, classical temporal logic seems not the right choice for us. Moreover, temporal logic is limited in its expressive power. Although we could introduce a version of temporal logic that talks about input and output of computations, we prefer to talk about the interface behaviour in terms of more general and more expressive interface assertions, given by predicates that contain the channel identifiers of the syntactic interface of a system as identifiers for streams. An interface assertion is a formula, which refers to the input and output channels of the systems as variables for timed streams. This way we write logical formulas that express properties of the input and output streams of hybrid systems. These formulas are written in classical predicate logic using, in addition, a number of basic operators for streams.

1.3.1 Events in Continuous Streams

A permanent continuous stream s is represented by a continuous function. The values of the function define the valuation of an attribute of the system at any chosen point in time. This defines for each time a kind of interface state. An *event* then can be defined as “a significant change in state”.

With a continuous stream s we associate a certain (logical) event at time t if s fulfils a particular property at time t. Simple examples would be that the continuous stream s has reached a particular value at time t, or assumes a maximum or a minimum at time t, or that its values in an interval around t lie in a certain range.

In full generality, an event e is a predicate

$$e : PTS \times \mathbb{R}_+ \to \mathbb{B}$$

We say that an event occurs at time t in the hybrid stream s if e(s, t) holds. Events provide a logical view onto hybrid streams. By definition there are lots of events. Which events are of relevance for a system has to be determined depending on the logics of the application. Typical examples would be "target speed reached", "temperature too high", "speed too high" or "signal available".

Actually an infinite number of events may occur in a given stream. Given a stream s, a set of events E and a time $t \in \mathbb{R}_+$ the set

$$\{t' \in [t : t+\varepsilon[: \forall e \in E : e(s, t')\}$$

is called the (t, ε)-*footprint* for an event set E on stream s. A (t, ε)-footprint may be *discrete*, *dense* or *durable*. Accordingly, we call an event *durable*, if it holds for all time points in some interval, which means that its footprint is identical to the set [t: t + ε[.

An event e is called *flickering at time* t in stream s, if one of the following formulas is valid:

(a) $\forall\, \varepsilon \in \mathbb{R}_+, \varepsilon > 0 : \exists\, t', t'' \in]t, t + \varepsilon[: e(s, t') \wedge \neg e(s, t'')$
(b) $\forall\, \varepsilon \in \mathbb{R}_+, \varepsilon > 0 : \exists\, t', t'' \in]t\text{-}\varepsilon, t[\cap \mathbb{R}_+ : e(s, t') \wedge \neg e(s, t'')$

In case (a) event e is called *flickering after* t, in case (b) *flickering before* t.

We say that an event is not Zeno, if it is never flickering. We say that for a stream a an event e is

- *Switched on* at time t, if $e(s, t) \wedge \forall \varepsilon \in \mathbb{R}_+, \varepsilon > 0 : \exists\, t' \in [t - \varepsilon, t[: \neg e(s, t')$
- *Switched off* at time t, if $e(s, t) \wedge \forall \varepsilon \in \mathbb{R}_+, \varepsilon > 0 : \exists\, t' \in [t, t + \varepsilon[: \neg e(s, t')$

and e is not flickering in s at time t.

Given event e, by ¬e we denote the complement event of e.

We call a set of events E ε-discrete for a stream s if for a real number $\varepsilon \in \mathbb{R}_+$ all (t, ε)-footprints for s and E contain at most one element. Then there is at most one event from the event set E in every time interval of length ε. In this case we can associate a discrete stream of events with the continuous stream s.

We are interested in associating a timed discrete stream of events with each hybrid stream to capture the event logics of histories. Given some time granularity $\delta \in \mathbb{R}_+$ we relate a discrete stream $r \in (E^*)^\infty$ with each timed stream $s \in PTS$ by

- Defining a set E of events
- Mapping the hybrid stream s to a discrete stream r = dis(s, E) by a function

$$dis : PTS \times E \to (E^*)^\infty$$

The set E is called the set of *logical observations*. To define the set of events E we assume a set of given events E_0. Since the set E_0 may not have discrete footprints and may contain durable events for streams $s \in PTS$ we replace durable events e by two events e_α and e_ω, where durable event e is starting at time t, characterizing the beginning and the end of the phase in which the event is durable by choosing and specifying:

$$e_\alpha(s,t) =_{\text{def}} \exists \varepsilon \in \mathbb{R}_+ \backslash \{0\} : \forall t' \in \mathbb{R}_+ \cap [t-\varepsilon : t[\ : \neg e(s,t')$$
$$\wedge \forall t' \in \mathbb{R}_+ \cap]t : t+\varepsilon[\ : e(s,t')$$
$$e_\omega(s,t) =_{\text{def}} \exists \varepsilon \in \mathbb{R}_+ \backslash \{0\} : \forall t' \in \mathbb{R}_+ \cap [t-\varepsilon : t[\ : e(s,t')$$
$$\wedge \forall t' \in \mathbb{R}_+ \cap]t : t+\varepsilon[\ : \neg e(s,t')$$

This gives us a set of events E' derived from E by replacing all durable events by the shown two events. This approach does work only, however, if the stream s avoids flickering and Zeno's paradox.

To avoid Zeno's paradox and generally flickering events we use a finite time granularity $\delta > 0$ and define the set of events E from E' as follows. We replace each event $e \in E'$ by an event e_E that is specified as follows. We make that sure we debounce events such that every event in E' can occur only once in each interval of length δ. To do this we first assume a linear strict order $<_{\text{prio}}$ defining priorities on the set of E' specifying the importance of events. Based on these priorities we define the event e_{prio} for each event e

$$e_{\text{prio}}(s,t) = (e(s,t) \wedge \neg \exists d \in E',\ t' \in \mathbb{R}_+ \cap \left[t - \frac{1}{2}\delta, t + \frac{1}{2}\delta\right] : d(s,t') \wedge e <_{\text{prio}} d)$$

i.e., e_{prio} (s, t) is an event with highest priority in the interval $\left[t - \frac{1}{2}\delta, t + \frac{1}{2}\delta\right]$. Now we define the set of events

$$E = \{e_{\text{prio}} : e \in E'\}$$

Given the set of events E we define a function

$$\text{dis}_E : PTS \to (E^*)^\infty$$

as follows:

$$\text{dis}_E(s)(i) = \langle\rangle \Leftarrow \neg \exists e \in E,\ t \in [i\delta : (i+1)\delta[: e(s,t)$$
$$\text{dis}_E(s)(i) = \langle e \rangle \Leftarrow \exists e \in E,\ t \in [i\delta : (i+1)\delta[: e(s,t)$$

Note that this definition is consistent since due to our construction every (t, δ)-footprint carries at most one event.

The time distance δ determines the time granularity of the stream $\text{dis}_E(s)$. If a finer or coarser time granularity is needed for $\text{dis}_E(s)$ the time granularity can be

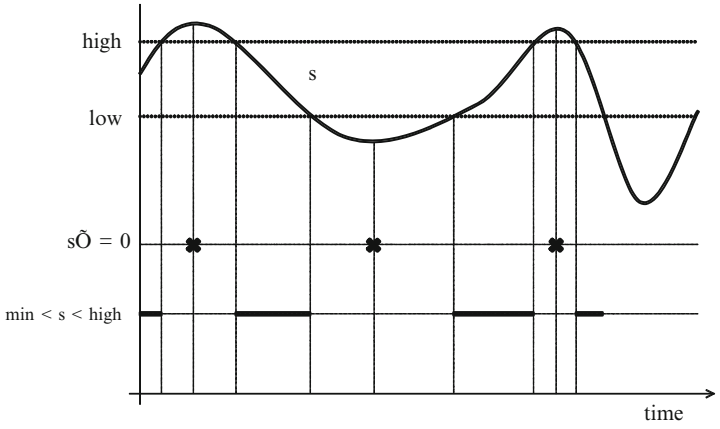

Fig. 1.3 Continuous stream s and discrete and durable events

changed according to [1]. Then after deriving $dis_E(s)$ we may coarsen $dis_E(s)$ which may lead to histories with more than one message in one time interval.

In principle, we may use the same construct to deal with durable events. Then durable events e are replaced by a sequence of discrete events e_E that are repeated every δ-time step as long as event e lasts. This is called *sampling* and provides an alternative to the approach treating durable events by introducing the event e_α and the event e_ω indicating the end of the durable event.

Fact 1.1. Associating discrete streams with hybrid streams

Let E′ be an arbitrary set of events and E be defined as above, then for every stream s ∈ PTS the set of events from E defines a discrete stream $dis_E(s) \in (E^*)^\infty$ of events in E.

Note that this form of associating discrete streams with continuous ones is essentially different from the techniques of discretisation used in numerical analysis or in control theory, where continuous functions are approximated by discrete step functions, where the distances between the discrete time points are chosen fine enough such that the functions are approximated precisely enough.

Figure 1.3 shows a continuous stream s and some examples of discrete and durable events.

Given event sets for all channels in set C, we get this way a function

$$\mathrm{Dis} : \vec{\vec{C}} \to \vec{C}$$

that maps histories of hybrid streams onto histories of discrete streams of events.

1.3.2 Assertions Specifying Hybrid Systems

To formulate properties about hybrid state machines with input and output we use interface assertions that refer to streams communicated via the input and output channels of the state machine. An interface assertion is a formula in predicate logic that contains the input and output channels of the state machines as logical identifiers for timed streams. The validity of such assertions for state machines is described in the following.

We work with templates to specify systems very much along the lines of [2] as demonstrated in the following example:

Example 1.1. Specification of hybrid systems

As a simple example we specify a hybrid system called Amplifier with a permanent and a discrete input stream and a permanent output stream.

System Amplifier (δ: Real: $\delta > 0$)

in v: Prm Real, c: Dsc Real
out r: Prm Real
$\forall t \in \mathbb{R}_+$:
$r(t + \delta) = lt(c, t)^* v(t)$
$0 \leq t < \delta \Rightarrow r(t) = 0$
where
$\forall t \in \mathbb{R}_+$:
$lt(c, t) = c(\max\{s" \in dom(c) : s" \leq t - \delta\}) \Leftarrow \{s" \in dom(c) : s" \leq t - \delta\} \neq \emptyset$
$lt(c, t) = 0 \Leftarrow \{s" \in dom(c) : s" \leq t - \delta\} = \emptyset$

This example shows an amplifier that amplifies the permanent input on channel v by the last actual value received on the discrete channel c and sends it as output on channel r with a time delay δ.

As the example shows, we use a mixture of plain higher order predicate logic and functional calculus. This leads to a specific logic that may be supported by interactive theorem provers (see [3]). Another possibility is domain specific logical calculi. Duration calculus (DC), for instance, is an interval logic for real-time systems (see [4]).

1.4 Composition

So far we have introduced a mathematical model for systems. Systems can be composed to larger systems by composition.

1.4.1 Composing Systems

In this chapter we study the composition of systems. We introduce the composition operator for composing two systems. Systems are composed by parallel composition with feedback following the approach of [2].

1.4.1.1 Composition of Systems in Terms of Their Interface Behaviour

The definition of composition of systems given by their interface behaviour reads as follows:

Definition 1.5. Composition of systems

Given two interfaces $F_1 \in IF[I_1 \blacktriangleright O_1]$ and $F_2 \in IF[I_2 \blacktriangleright O_2]$, with type consistent channels and where $O_1 \cap O_2 = \emptyset$, we define a composition for the feedback channels $C_1 = O_1 \cap I_2$ and $C_2 = O_2 \cap I_1$ by the expression

$$F1 \otimes F2$$

The system $F_1 \otimes F_2 \in IF[I \blacktriangleright O]$ is defined as follows (let $C = I_1 \cup O_1 \cup I_2 \cup O_2$, where $I = (I_1 \backslash C_2) \cup (I_2 \backslash C_1)$ and $O = (O_1 \backslash C_1) \cup (O_2 \backslash C_2)$):

$$\forall x \in \vec{I} : (F1 \otimes F2)(x) = \{y \in \vec{O} : \exists z \in \vec{C} : y = z|O$$
$$\wedge x = z|I \wedge z|O_1 \in F1(z|I_1) \wedge z|O_2 \in F_2(z|I_2)\}$$

The channels in set $C_1 \cup C_2$ are called *internal* for the composed system $F_1 \otimes F_2$. □

The idea of the composition of systems as defined above is graphically illustrated in Fig. 1.4.

In a composed system $F_1 \otimes F_2$, the channels in the channel sets C_1 and C_2 are used for internal communication.

Given specifying assertions S_1 and S_2 for the systems F_1 and F_2, the specifying assertion for $F_1 \otimes F_2$ is given by the assertion $\exists\, C_1, C_2 \colon S_1 \wedge S_2$, where internal channels C_1 and C_2 are hidden by the existential quantifier.

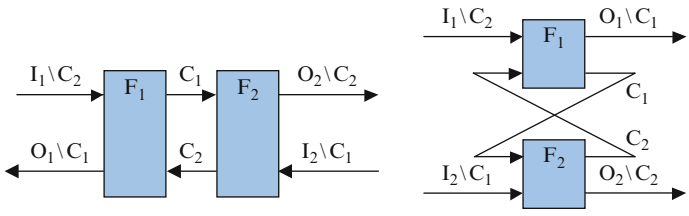

Fig. 1.4 Composition $F_1 \otimes F_2$ (in two layouts)