TUM

INSTITUT FÜR INFORMATIK

Mini-Statecharts: A Lean Version of Statecharts

Dieter Nazareth
Franz Regensburger
Peter Scholz

TUM-I9610
Februar 1996

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-02-1996-I9610-350/1.-FI

Typescript: ---

Druck: Mathematisches Institut und
Institut für Informatik der
Technischen Universität München

Mini-Statecharts: A Lean Version of Statecharts*

Dieter Nazareth, Franz Regensburger, Peter Scholz

Technische Universität München, Institut für Informatik
D-80290 München, Germany
E–Mail: {nazareth,regensbu,scholzp}@informatik.tu-muenchen.de

*This work is partially sponsored by the German Federal Ministry of Education and Research (BMBF) as part of the compound project "KorSys" and by BMW (Bayerische Motoren Werke AG).

Abstract

Statecharts are well accepted in industrial applications for specifying reactive, embedded systems. Unfortunately, a reference semantics has not been developed until now. Therefore, the semantics of Statecharts is still of interest in the science community. This paper presents a compositional, denotational semantics for a special subclass of Statecharts based on stream processing functions. The basic component of Mini-Statecharts is a deterministic, reactive, signal-triggered sequential automaton which can be composed in multiple ways. The composition operators are: parallel composition, local signal-scoping, semantic feedback of signals, and decomposition of states. The main issues are the compositionality of the semantics, the definition of the semantic behavior for a kind of history mechanism, and the different views of the feedback operator.

Contents

1 Introduction

Statecharts [Har87] are a visual specification language proposed for specifying reactive systems. They extend conventional state transition diagrams with structuring and communication mechanisms. These mechanisms enable the description of large and complex systems. Due to this fact Statecharts have become quite successful in industry. The full Statecharts language, however, contains many mechanisms that cause problems concerning both syntax and semantics. A good description of these problems can be found in [vdB94].

In this paper, we describe a restricted version of Statecharts, called *Mini-Statecharts*. We have adopted the approach taken by Argos [Mar92]. While Mini-Statecharts are powerful enough to describe large and complex reactive systems, nevertheless, we can assign a concise, formal semantics to them. The most important restriction is that we do not allow *inter-level* transitions. These are transitions between nodes on different levels. This means that they cross the borderline of one or more states. Additionally, we disallow references to state names, i.e., events like $entered(\sigma)$, $entering(\sigma)$ for any state σ. Inter-level transitions and state references impede the definition of a compositional semantics. Both mechanisms can be substituted by communicating appropriate signals. As a consequence, Statecharts cannot be developed in a *modular* way. Mini-Statecharts, however, are clearly decomposed into Sub-Mini-Statecharts. Thus, they can be constructed by simply sticking them together.

We present a compositional, denotational semantics for Mini-Statecharts. In particular, we concentrate on different semantic views of the feedback operator which is responsible for signal communication. These views are in particular *instantaneous, macro-/micro-step, and delayed feedback.* These feedback operators are simultaneously available in our language. Furthermore, unlike [Mar92, HRdR92, HL95], we have formalized the behaviour of *history* refined states, but we prohibit history entrances for special transitions into a sub-chart. History functionality is considered as an attribute of an entire chart, not of a single transition. Either *all* transitions into a sub-chart are history transitions or *none* of them.

The semantics of Mini-Statecharts is given in a fully functional way. It can be immediately executed by a suitable interpreter. Thus, we do not only define a theoretical semantics, but in addition provide a simple program for simulating and prototyping Mini-Statecharts. This is in contrast to existing tools like *Statemate* [Har90, Inc90], where the semantic behavior of the prototyping tool sometimes differs from published Statecharts semantics. In our approach there exists exactly one semantics. It can be used to prototype reactive systems as well as to reason about systems in a suitable theorem prover, like *Isabelle* [Pau94].

The rest of the paper is structured as follows. In Section 2 we informally introduce Mini-Statecharts and present a formal, abstract syntax for them. Section 3 starts with a brief introduction to streams and stream processing functions. We develop a denotational, compositional step semantics for Mini-Statecharts and lift it to a semantics based on stream processing functions. Then we add the delayed feedback operator. The section concludes with a comparison of the different feedback operators. Finally, in Section 4 we give a brief conclusion and discuss future extensions.

2 Abstract Syntax

In this section we propose a textual syntax for Mini-Statecharts. We give an abstract, inductively defined syntax for the set of all Mini-Statecharts $\mathcal{S}$. Let M denote a (potentially infinite) set of signal-names, *States* a (potentially infinite) set of state-names and $\mathcal{B}(M)$ the Boolean terms [Bro93] over M. $\wp_{fin}(X)$ denotes the set of finite subsets of some set X. Now we can formally describe our abstract syntax.

2.1 Sequential Automata

Sequential automata are the basic elements of Mini-Statecharts. The construct

$$(\Sigma, \sigma_d, \sigma, \delta)$$

is an element of $\mathcal{S}$ iff the following syntactic constraints hold:

1. $\Sigma \in \wp_{fin}(States)$ denotes the nonempty finite set of all states of the automaton.

2. $\sigma_d, \sigma \in \Sigma$ represent the default state and the actual state, respectively. In contrast to conventional sequential automata we do actually need the state σ_d to initialize Mini-Statecharts for reentering non-history decomposed states (see Section 3.2).

3. $\delta : \Sigma \times \mathcal{B}(M) \to \Sigma \times \wp_{fin}(M)$ is the finite, partial state transition function that takes a state and a Boolean term and yields the subsequent state together with a finite set of output signals. For every Boolean variable $a \in M$ in the term $t \in \mathcal{B}(M)$ the occurrence of a means that signal a has to be present and $\neg a$ means that this signal has to be absent to enable the trigger condition. Of course, we also allow Boolean terms like $\neg(a \wedge b)$. In this case, both signals must not be present to enable the condition. Trigger conditions formulated over Boolean terms allow any combination of absent or present signals as guard.

Obviously, we have chosen a very abstract syntactic notation for automata. This choice was made to keep the semantic definitions easily understandable. In a more concrete syntax the finite transition function could be represented by a tabular-like or graphic notation. As the reader may already have detected, we do not explicitly denote the set of signals that the automaton $A = (\Sigma, \sigma_d, \sigma, \delta)$ can react on. This set is implicitly given by the transition function δ. δ is exactly defined for these signals that A can react on.

Example 1 (Sequential Automaton) *We want to introduce our syntax by the aid of an example which is borrowed and adapted from [HdR91]. It is a television set with remote control. Only two programs (1 and 2) can be received. Input events are provided by pressing the buttons "on", "off", "txt", "sound", "mute", "1", and "2" on the remote control unit. The Mini-Statechart that describes switching the TV on and off is graphicly described in Fig. 1. The graphic notation is borrowed from [Har87]: the default state is characterized by an extra arrow and every transition between states σ and σ' is labeled with "t/x", iff $\delta(\sigma, t) = (\sigma', x)$. The textual version of Fig. 1 is defined as follows:*

$$S_{TV} = (\{\mathrm{ON}, \mathrm{STANDBY}\}, \mathrm{ON}, \mathrm{ON}, \delta_{TV})$$

where the partial function δ_{TV} is given by:

$$\begin{aligned} \delta_{TV}(\mathrm{ON}, \mathrm{off}) &= (\mathrm{STANDBY}, \{\}) \\ \delta_{TV}(\mathrm{STANDBY}, \mathrm{on}) &= (\mathrm{ON}, \{\}). \end{aligned}$$

$\{\}$ *means that no signals are generated at all.*

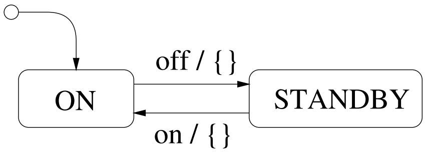

Figure 1: Sequential Automaton

2.2 Parallel Composition

In this section we introduce parallel composition, capturing the property that, staying in a state, the system has to stay in *all* of its parallel components [Har87]. Parallel components may be considered to be orthogonal. Suppose S_1 and S_2 are elements of the set $\mathcal{S}$ of Mini-Statecharts. Then their parallel composition denoted by the syntax

$$\mathsf{And}\ (S_1, S_2)$$

is in $\mathcal{S}$, too. There are no syntactic restrictions on this composition. This leads to a Mini-Statechart that behaves like S_1 and S_2 simultaneously. In the graphic notation parallel components are separated by splitting a box into components using dashed lines [Har87]. Being in a parallel component means being in all of its substates at the same time, independently and concurrently. Note that the pure parallel composition does not contain any broadcast communication mechanism as in the original literature. Communication is carried out explicitly by the aid of our feedback operators which will be introduced in Section 3.2.2.

Example 2 (Parallel Composition) *In a TV, the operations of sound and image are independent from each other, i.e., switching with the "txt" button from normal mode to videotext mode does not affect the sound, et vice versa. For simplicity, we have only two sound levels, "MUTE" and "ON". The parallel Mini-Statechart that formally describes this behavior is denoted in the following and graphicly represented in Fig. 2.*

$$\mathsf{And}\ (S_{IMAGE}, S_{SOUND})$$

where the abbreviations S_{IMAGE} and S_{SOUND} are denoted as follows:

$$S_{IMAGE} = (\{\mathrm{NORMAL}, \mathrm{VIDEOTEXT}\}, \mathrm{NORMAL}, \mathrm{NORMAL}, \delta_{IMAGE})$$

where

$$\begin{aligned} \delta_{IMAGE}(\mathrm{NORMAL}, \mathrm{txt}) &= (\mathrm{VIDEOTEXT}, \{\}) \\ \delta_{IMAGE}(\mathrm{VIDEOTEXT}, \mathrm{txt}) &= (\mathrm{NORMAL}, \{\}) \end{aligned}$$

$$S_{SOUND} = (\{\mathrm{MUTE}, \mathrm{SOUNDON}\}, \mathrm{MUTE}, \mathrm{MUTE}, \delta_{SOUND})$$

where

$$\begin{aligned}\delta_{SOUND}(\mathrm{MUTE}, \mathrm{sound}) &= (\mathrm{SOUNDON}, \{\})\\ \delta_{SOUND}(\mathrm{SOUNDON}, \mathrm{mute}) &= (\mathrm{MUTE}, \{\}).\end{aligned}$$

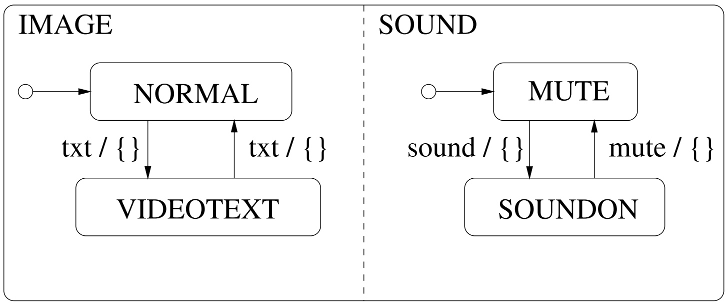

Figure 2: Parallel Composition

2.3 Hierarchical Decomposition

Besides orthogonality, depth is another important feature of Statecharts. The concept of hierarchically structuring the state space is essential for Statecharts. Hence, we have adapted the possibility of specifying hierarchically decomposed charts in our approach. Hierarchical decomposition is applied to express the refinement of the decomposed state. Suppose that $(\Sigma, \sigma_d, \sigma, \delta) \in \mathcal{S}$ is a sequential automaton. Then

$$\mathsf{Dec}\ (\Sigma, \sigma_d, \sigma, \delta)\ \mathsf{by}\ \varrho$$

is in $\mathcal{S}$, too, iff:

$$\varrho : \Sigma \rightarrow (\mathcal{S} \times \{\mathsf{History}, \mathsf{NoHistory}\}) \cup \{\mathsf{NoDec}\}$$

is a total, finite function. With respect to the construct $\mathsf{Dec}\ (\Sigma, \sigma_d, \sigma, \delta)\ \mathsf{by}\ \varrho$ the sequential automaton $(\Sigma, \sigma_d, \sigma, \delta)$ is called the *master*. A state $\sigma \in \Sigma$ with $\varrho(\sigma) \neq \mathsf{NoDec}$ (where NoDec stands for *no decomposition*) is called a *refined* state of the master whereas $S := \pi_1(\varrho(\sigma))$ is called the *slave* of the master which is controlled by state σ. π_i denotes the i-th projection.

The effect of this decomposition can be described by the following rules. Whenever the current state of the master is σ and $\varrho(\sigma) = \mathsf{NoDec}$, then $\mathsf{Dec}\ (\Sigma, \sigma_d, \sigma, \delta)\ \mathsf{by}\ \varrho$ has a behavior according to $(\Sigma, \sigma_d, \sigma, \delta)$. Otherwise, when σ is entered, $\mathsf{Dec}\ (\Sigma, \sigma_d, \sigma, \delta)\ \mathsf{by}\ \varrho$ starts behaving like master and slave simultaneously. When σ is left, we distinguish between *preemptive* and *non-preemptive* exit/interrupt.

With a non-preemptive interrupt, the slave first terminates its action concerning the current input signals and then is left. The preemptive interrupt immediately interrupts the slave and abruptly terminates its action. Due to this behavior, we say that the transition on the higher level of hierarchy has also a higher level of *priority*. Unlike [Mar92], all information about the current state(s) of the slave can be stored, iff $\pi_2(\varrho(\sigma)) = \mathsf{History}$. In the case that $\pi_2(\varrho(\sigma)) = \mathsf{NoHistory}$, S is reinitialized.

In contrast to many other approaches, for example [Mar92, HRdR92], this paper provides a formal semantics for *history decomposed* states. In our approach, we consider a state σ to be history decomposed, iff $\pi_2(\varrho(\sigma)) = \mathsf{History}$. When entering a “normal”, i.e., non-history decomposed state σ ($\pi_2(\varrho(\sigma)) = \mathsf{NoHistory}$), all sequential automata of the slave are entered by their default states. However, if σ is a history decomposed state, we have to distinguish two different cases. If σ was never entered before, all sequential automata of the slave are entered by their default states as is in the non-history case. Otherwise, the sequential automata of the slave are entered by the states most recently visited, i.e., by their actual states. Generally, history is applied only on the level in which it appears. If the history mechanism is made to apply all the way down to the lowest level of states, this is called *deep history* in [Har87]. In this paper we only explain a formal semantics for deep history decomposed states.

Mini-Statecharts conclude a clear and effective way to express hierarchical structures. In contrast to Statecharts, this decomposition is fully modular because we prohibit *inter-level transitions*. Inter-level transitions are transitions between states of different levels of hierarchy. An example is pictured in Fig. 3. Prohibition of inter-level transitions implies that it is impossible to leave a master in dependence on the current state(s) of its slave. In Section 3.2.2 we will show how to bypass this problem.

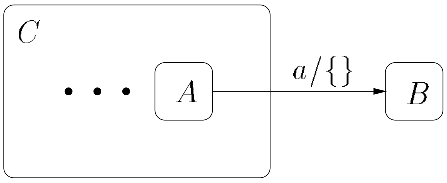

Figure 3: Inter-level Transition

Example 3 (Hierarchical Decomposition) *If we take a closer look at Fig. 1 and Fig. 2 we find out that the latter one is a decomposition of the state “ON”. When the TV is in state “ON”, it can be either in normal mode or in videotext mode* and *the sound can be on or off. The overall diagram is shown if Fig. 4. Staying in state “ON”, the outer transition which is triggered by “off” has a higher priority than the inner ones, for example, the ones labeled with “txt”. If we assume that both signals “off” and “txt” are simultaneously present — this means that both buttons have been pressed — the system first changes the internal state of “ON” and then leaves it. This semantic property, which is a kind of non-preemptive or weak interrupt, is formally denoted in the subsequent section. The formal syntax for the decomposition of the state “ON” reads:*

$$\mathsf{Dec}\ S_{TV}\ \mathsf{by}\ \varrho_{TV}$$

where the partial function ϱ_{TV} is defined in the sequel:

$$\begin{aligned}\varrho_{TV}(\mathrm{ON}) &= (S_{ON}, \mathsf{NoHistory})\ \textit{where}\\ &\quad S_{ON} = \mathsf{And}\ (S_{IMAGE}, S_{SOUND})\\ \varrho_{TV}(\mathrm{STANDBY}) &= \mathsf{NoDec}.\end{aligned}$$

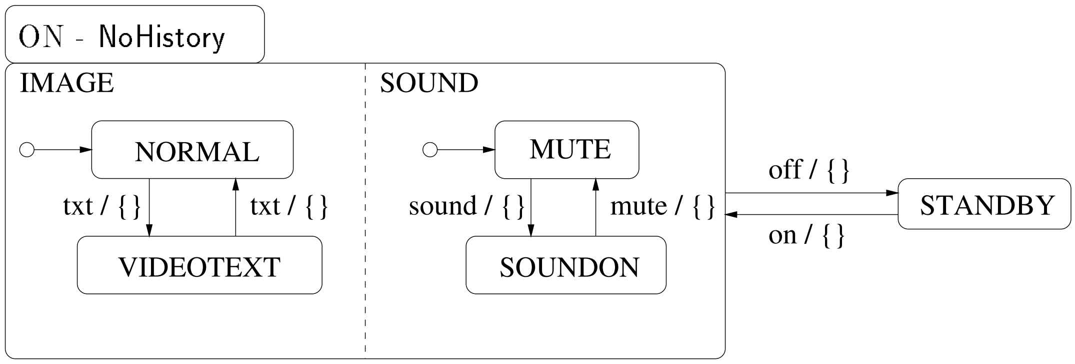

Figure 4: Hierarchical Decomposition

2.4 Feedback of Certain Signals

Parallel composition is used to denote orthogonal components. However, mostly, parallel components are not fully orthogonal. Therefore, Statecharts provide a broadcast communication mechanism to pass messages between components working in parallel. In [Har87] this behavior already is integrated in the orthogonal composition of Statecharts. Broadcasting is achieved by feeding back all generated signals to all components. This means that there exists an *implicit* feedback mechanism at the outermost level of a Statechart. Unfortunately, this implicit signal broadcasting leads to a non-compositional semantics. We avoid this problem by adding an *explicit* feedback operator. In the literature different semantic views of the feedback mechanism can be found [vdB94]. Hence, we provide three different feedback operators to explore the practical usefulness of the most interesting views. Suppose that S is in $\mathcal{S}$ and $L \in \wp_{fin}(M)$ is the set of signals which should be fed back, then the constructs

$$\textsf{I-Feedback}\ (S, L), \quad \textsf{M-Feedback}\ (S, L), \quad \text{and} \quad \textsf{D-Feedback}\ (S, L)$$

are also in $\mathcal{S}$. They denote the instantaneous, the macro-/micro-step, and the delayed feedback, respectively. These operators differ in their signal propagation mechanisms: I-Feedback and D-Feedback feed the signals back at the same instant of time and at the next instant of time, respectively. M-Feedback is more complicated and therefore will be explained in detail in Section 3.2.2. There are no syntactic restrictions on these compositions.

Note that the feedback compositions can be combined with signal scoping, which will be presented in the subsequent section. Using feedback and hiding together makes the communication invisible for the environment of the components. Signals in L are called *internal signals* (relative to S). All other signals are called *external signals* (relative to S).

Example 4 (Feedback) *This example illustrates the feedback operator as well as the hierarchical history decomposition. We concentrate on the instantaneous feedback* I-Feedback *here. We assume that our TV has a really poor program offer which only consists of two channels. The state "NORMAL" (see Fig. 4) is decomposed as shown in Fig. 5. We need a two step decomposition to denote that only the "left" chart of the parallel component is history decomposed. The feedback operator is pictured in Fig. 5 as an extra box, sticked* *on the bottom of the Mini-Statechart.*

When we change from one channel to another, usually the sound is turned off for a moment, probably to avoid unwanted noises. To model this, we add two parallel components $S_{CHANNELS}$ *and* S_{SM} *(for switching mode). Pressing a channel button "1", "2" on the remote control, the internal signal "sm" is simultaneously generated and the TV switches to the corresponding channel. At the moment, the reader must be content with this informal explanation. We will give a precise definition of our time hypothesis in Section 3. "sm" is instantaneously fed back by the aid of* I-Feedback*. Therefore, the parallel automaton* S_{SM} *also is immediately triggered, i.e., reacts on the signal "sm" and simultaneously generates the signal "mute". If "mute" is fed back in Mini-Statechart* S_{ON}*, pictured in Fig. 4,* S_{SOUND} *now reacts on "mute". Finally, the sound will be turned off. After one time tick, the event "sound" is generated to turn it on again. Therefore, besides "mute" also "sound" has to be fed back in* S_{ON}*.*

$$\textsf{Dec } S_{IMAGE} \textsf{ by } \varrho_{IMAGE} \textit{ where}$$

$$\begin{array}{l}
\varrho_{IMAGE}(\mathrm{VIDEOTEXT}) = \textsf{NoDec} \\
\varrho_{IMAGE}(\mathrm{NORMAL}) = (S_{FNORMAL}, \textsf{NoHistory}) \textit{ where} \\
\quad S_{FNORMAL} = \textsf{I-Feedback } (S_{NORMAL}, \{\mathrm{sm}\}) \textit{ where} \\
\quad\quad S_{NORMAL} = \textsf{And } (\textsf{Dec } S_C \textsf{ by } \varrho_C, S_{SM}) \textit{ where} \\
\quad\quad S_C = (\{\mathrm{CH}\}, \mathrm{CH}, \mathrm{CH}, \varepsilon^1) \\
\quad\quad \varrho_C(\mathrm{CH}) = (S_{CHANNELS}, \textsf{History}) \textit{ where} \\
\quad\quad\quad S_{CHANNELS} = (\{\mathrm{CH1}, \mathrm{CH2}\}, \mathrm{CH1}, \mathrm{CH1}, \delta_{CHANNELS}) \textit{ where} \\
\quad\quad\quad\quad \delta_{CHANNELS}(\mathrm{CH1}, 1) = (\mathrm{CH1}, \{\mathrm{sm}\}) \\
\quad\quad\quad\quad \delta_{CHANNELS}(\mathrm{CH1}, 2) = (\mathrm{CH2}, \{\mathrm{sm}\}) \\
\quad\quad\quad\quad \delta_{CHANNELS}(\mathrm{CH2}, 1) = (\mathrm{CH1}, \{\mathrm{sm}\}) \\
\quad\quad\quad\quad \delta_{CHANNELS}(\mathrm{CH2}, 2) = (\mathrm{CH2}, \{\mathrm{sm}\}) \\
\quad\quad S_{SM} = (\{\mathrm{SILENT}, \mathrm{LOUD}\}, \mathrm{SILENT}, \mathrm{SILENT}, \delta_{SM}) \textit{ where} \\
\quad\quad\quad \delta_{SM}(\mathrm{LOUD}, \mathrm{sm}) = (\mathrm{SILENT}, \{\mathrm{mute}\}) \\
\quad\quad\quad \delta_{SM}(\mathrm{SILENT}, \neg\mathrm{sm}) = (\mathrm{LOUD}, \{\mathrm{sound}\}).
\end{array}$$

As we can see now, "mute" and "sound" are as well as "sm" internal signals. These signals have to be fed back relative to S_{ON}*. Therefore, we have to modify* ϱ_{TV}*:*

$$\begin{array}{l}
\varrho_{TV}(\mathrm{ON}) = (S_{FON}, \textsf{NoHistory}) \textit{ where} \\
\quad\quad S_{FON} = \textsf{I-Feedback } (S_{ON}, \{\mathrm{sound}, \mathrm{mute}\}).
\end{array}$$

Note that "sound" and "mute" can occur as internal and as external signals, respectively. Thus, we do not hide these signals and therefore do not use the scoping mechanism. Nevertheless, "sm" has to be hidden, because it is a pure internal signal which can never be generated by pressing a button on the remote control. Local hiding of signals will be presented in the following.

2.5 Local Hiding of Signals

Specifying large reactive systems possibly leads to large charts with many signal names. This may promote name clashes which could be avoided by the utilization of local hiding

[1] ε denotes the empty, i.e., totally undefined function.

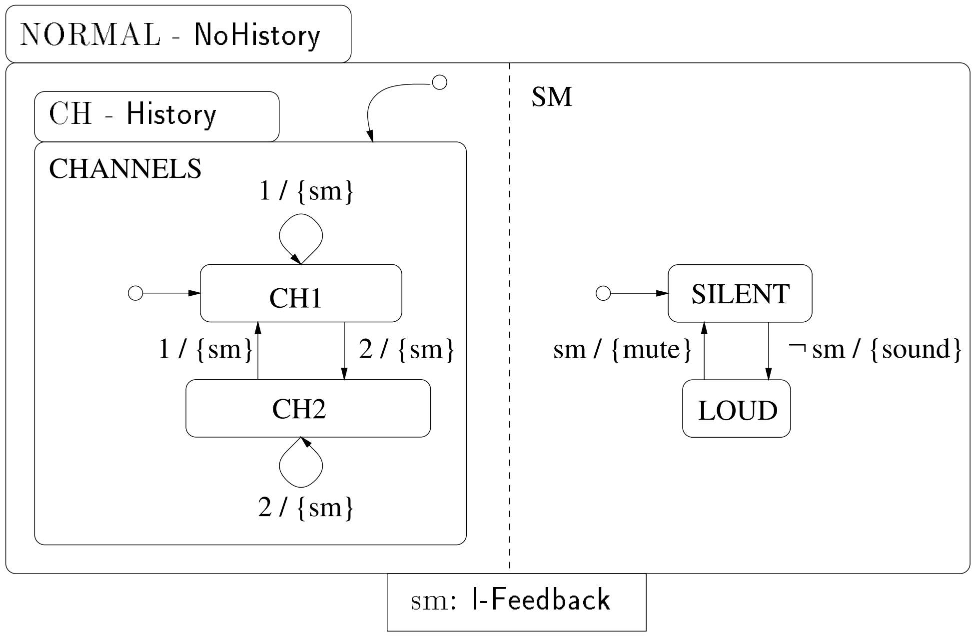

Figure 5: Feedback and Hierarchical Decomposition

of signals. It can be compared with the declaration of local procedure variables in a high-level programming language. Thus, to achieve modularity incoming and outgoing signals of a chart may be hidden. Suppose that S is in $\mathcal{S}$ and $L \in \wp_{fin}(M)$, then the construct

$$\mathsf{Local}\ (S, L)$$

is in $\mathcal{S}$, too. There are no syntactic restrictions on this composition. $\mathsf{Local}\ (S, L)$ hides any generation of any $l \in L$ by S and makes S insensitive to any l generated by the environment. Note that this operator is not available in conventional Statecharts. However, in our opinion it is essential to describe large reactive systems. It can be used to restrict signals to certain components of the system. In the graphic notation signal hiding is – like feedback – represented by an extra box at the bottom of the original chart.

Example 5 (Signal Hiding) *In Example 4 we pointed out that we have to hide signal "sm". Thus, we must modify $S_{FNORMAL}$:*

$$S_{FNORMAL} = \mathsf{Local}\ (\mathsf{I\text{-}Feedback}\ (S_{NORMAL}, \{\mathrm{sm}\}), \{\mathrm{sm}\})$$

Fig. 6 depicts the situation.

3 Semantics

3.1 Semantic Model: Streams and Stream Processing Functions

In this section we briefly discuss the notion of streams and stream processing functions which we will use throughout this paper. First of all there is no unique notion of streams in the literature. Our approach derives from domain theory and exploits the techniques

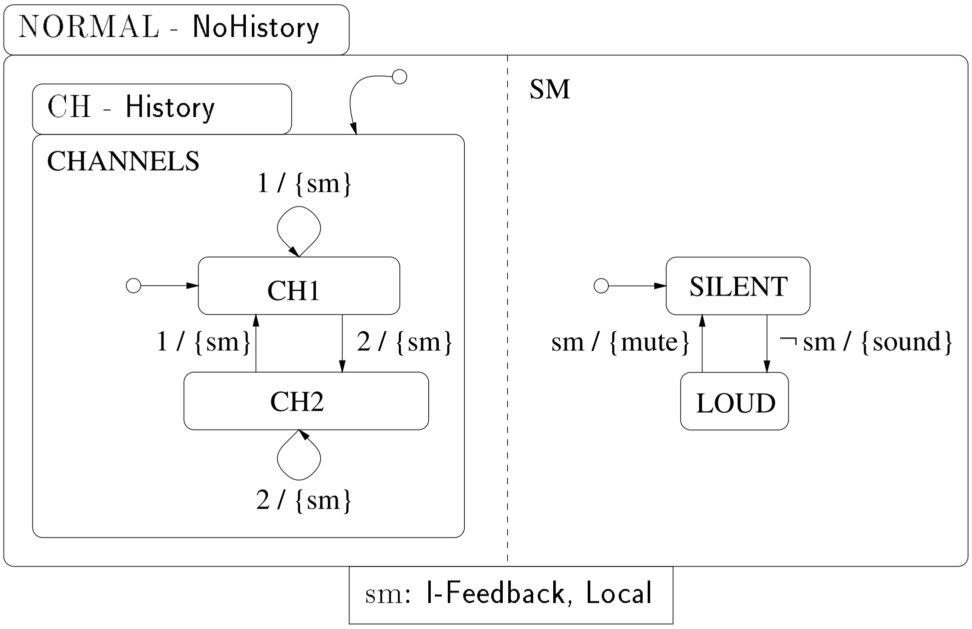

Figure 6: Signal Hiding

from [Pau87, Fre90, Gun92]. In our setting streams over the pcpo (pointed complete partial order) M are the initial solution to the domain equation

$$X \simeq M \otimes (X_\perp)$$

where $\otimes$ denotes the strict product of pcpo's and $X_\perp$ denotes the lifting of pcpo X. A brief introduction to the specification of distributed systems using the Focus methodology can be found in [SS95]. For a detailed description of Focus we refer to [BDD+93]. However, in this section we will give a more operational and informal explanation of the type of streams.

The type of streams we use throughout this paper is a list-like type. Lists are well known from functional programming. In the literature three different kinds of lists are distinguished: finite or strict lists, sequences and lazy lists. Our streams would be called sequences in [Pau92].

When we compute values in a programming language there is the possibility that the computation does not terminate. In our logical theory, there is a special denotation for this case. We use the symbol $\perp$ to indicate that a computation did not (yet) terminate[2]. Of course our logic, which is closely related to LCF [Pau87], has to respect this special constant. If we apply an operation f to $\perp$, which means to the result of a computation which did not (yet) terminate, we have do decide how much output f can produce from this lack of information. In a setting where operations have to evaluate all of their arguments before they start to compute, the result of applying f to $\perp$ would again yield $\perp$. However, in every reasonable programming language there is at least one operation which does not evaluate all of its arguments. This operation is the conditional. The conditional is called strict in its first and non-strict in its second and third argument.

[2]For a background discussion, the interested reader is referred to [Sto77] and several references therein to work by Dana Scott.

In most functional programming languages there are further non-strict operations. The basis for these non-strict operations are the constructors introduced in the definition of the types they operate on. In the special case of list-like types the central constructor is the so-called 'cons operator' for lists.

Throughout this paper we use M^ω as a notation for the type of streams over M. Our notation for the cons operator is &. Given an element m of type M and a stream s over M, the term $m\&s$ denotes the stream which starts with the element m followed by the stream s. When constructing the new stream with & we only insist in the definedness of m. Definedness means here, that m must be different from $\perp$. However, we are not interested in the definedness of s. This means that the cons operator & is strict in its first and non-strict in its second argument. Operationally, only the first argument is evaluated for the process of construction. More precisely, every stream $s \in M^\omega$ is either $\perp$ or is constructed by & from unique $m \in M$ and $s' \in M^\omega$ where m is different from $\perp$.

The destructor *ft* selects the first element of a stream. When applied to $m\&s$, where m is different from $\perp$, it yields m. Otherwise the result is $\perp$. The destructor *rt* selects the rest of a stream. When applied to $m\&s$ it yields s regardless of the definedness of s. Otherwise the result is $\perp$.

The constructor & and the destructors *ft* and *rt* together with a fixed point operator are basically all the operations we need to formulate other operations on streams including stream processing functions. However, throughout this paper we use equational systems together with local definitions (let terms) to formulate recursive operations on streams.

Besides the constructor and the destructors we use an auxiliary function $s \downarrow i$ which yields for a positive natural number i the i-th element of stream s. The function $\downarrow$: $M^\omega \times PNat \to M$ is strict in both arguments and has the following logical properties on defined arguments:

$$\begin{aligned} m\&s\downarrow 1 &= m \\ m\&s\downarrow(i+1) &= s\downarrow i. \end{aligned}$$

By the aid of this technical background we should be able to understand the stream semantics for Mini-Statecharts presented in Section 3.3. This stream semantics is derived from a stepwise denotation which is going to be developed in the following section.

3.2 The Step Semantics

The underlying time hypothesis of our semantics is a global time measure. We assume that every Mini-Statechart can make a step — at least an idle loop — at every single time tick. This assures time progress because every single transition takes place in exactly one time unit [GS95]. This behavior is guaranteed by reactive automata. In this section we are not able to deal with the delayed feedback operator $\mathsf{D\text{-}Feedback}$. The reason for neglecting it is obvious: even without having yet formally explained the delayed feedback operator the reader can easily imagine that when only talking about one single, namely the *actual* step it is impossible to make conclusions about the *next* step. The functionality of the strict step function is:

$$st[\![.]\!] : \mathcal{S}_{\mathsf{D}} \to \wp_{fin}(M) \to \wp_{fin}(M) \times \mathcal{S}_{\mathsf{D}}$$

where $\mathcal{S}_{\mathsf{D}}$ is defined by $\mathcal{S}$ without the D-Feedback operator. The step semantics yields a finite set of output signals together with the subsequent Mini-Statechart which is necessary for defining the step semantics of the history operator as well as for the overall stream semantics. The rest of this section defines the step semantics for each element of the syntactic category of $\mathcal{S}_{\mathsf{D}}$.

3.2.1 Composition

Sequential Automaton

Informally, a sequential, deterministic and reactive automaton $(\Sigma, \sigma_d, \sigma, \delta)$ takes a set of input signals, the so-called *stimuli*, produces a set of signals as output and then behaves like an automaton with modified actual state. Before we formally describe the step semantics of this kind of automaton, we have to do some observations first.

The transition function δ is worth of a more detailed consideration. δ is defined on Boolean terms. Reactive systems, however, have to react to a set of signals. Thus, we have to define which transition is triggered by a given set of signals. For this reason, we use a strict and total function *trigger* interpreting a Boolean term over signals with respect to some given set of signals:

$$\mathit{trigger}\ : \mathcal{B}(M) \times \wp_{fin}(M) \rightarrow \{\mathit{tt}, \mathit{ff}, \perp\}.$$

Remember that for every Boolean variable $a \in M$ in term $t \in \mathcal{B}(M)$ the occurrence of a means that signal a has to be present and $\neg a$ means that this signal has to be absent to enable the trigger condition. Because $(\wedge, \neg)$ is a possible basis for Boolean terms we define *trigger* for these constructs only. If one wants to deal with $\vee, \Rightarrow, \Leftrightarrow$, etc., *trigger* simply has to be adapted in a straight forward fashion. Let $a \in M$, $x \in \wp_{fin}(M)$ and $t, t_1, t_2 \in \mathcal{B}(M)$ then

$$\begin{array}{rcl} \mathit{trigger}\ (a, x) & := & a \in x \\ \mathit{trigger}\ (t_1 \wedge t_2, x) & := & \mathit{trigger}\ (t_1, x) \text{ and } \mathit{trigger}\ (t_2, x) \\ \mathit{trigger}\ (\neg t, x) & := & \text{not } \mathit{trigger}\ (t, x). \end{array}$$

To get a semantics which deals with sets of signals instead of Boolean terms, in the sequel, we consider a total, deterministic state transition function δ' with the functionality

$$\delta' : \Sigma \times \wp_{fin}(M) \rightarrow \Sigma \times \wp_{fin}(M).$$

For $\sigma \in \Sigma_A$ and $x \in \wp_{fin}(M)$ we define:

$$\delta'(\sigma, x) := \begin{cases} \delta(\sigma, t) & \text{if } \exists t \in \mathcal{B}(M), \sigma' \in \Sigma, y \in \wp_{fin}(M) : \delta(\sigma, t) = (\sigma', y) \wedge \\ & \mathit{trigger}\ (t, x) = \mathit{tt} \\ (\sigma, \{\}) & \text{else.} \end{cases}$$

Note that the function δ is only defined for finitely many $t \in \mathcal{B}(M)$. Therefore, the above existential quantifier is easily decidable. Obviously, δ' is a total function. Every sequential automaton with a total state transition function is *reactive* which means that it can make a step at every single time tick. This represents the characterizing property of reactive systems. Additionally, we require *deterministic* automata which is expressed by[3]:

$$\forall x \in \wp_{fin}(M), \sigma \in \Sigma : \exists_1 t \in \mathcal{B}(M), \sigma' \in \Sigma, y \in M :$$
$$\delta(\sigma, t) = (\sigma', y) \wedge \mathit{trigger}\ (t, x) = tt.$$

This property ensures δ' to be a well-defined function. Besides simulation, *Statemate* [Inc90], a Statecharts tool, provides the opportunity to generate executable, deterministic C code. The non-determinism in a Statemate specification is resolved by the aid of complicated rules. Therefore, we have decided to focus upon a deterministic semantics right from the beginning. However, from a theoretical point of view there is no difficulty to handle nondeterministic sequential automata.

The behavior of a sequential automaton is denoted by:

$$\begin{aligned} &st[\![(\Sigma, \sigma_d, \sigma, \delta)]\!]x = \\ &\qquad \text{let } (\sigma', y) = \delta'(\sigma, x) \\ &\qquad \text{in } \ (y, (\Sigma, \sigma_d, \sigma', \delta)). \end{aligned}$$

Note that the default state σ_d is never changed. In the sequel we will picture the first projection of the step semantics, i.e., the signal-flow as (hierarchical) data-flow network. For the sequential automaton this is shown in Fig. 7[4].

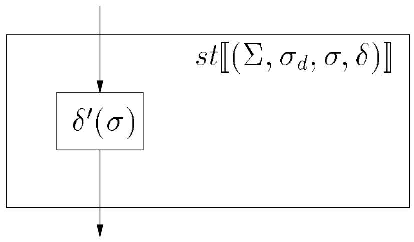

Figure 7: Sequential Automaton

Parallel Composition

Informally, the orthogonal composition of Statecharts behaves as S_1 and S_2 synchronously together. Generated signals of the parallel components are unified. As a consequence, multiple occurrences of one signal are neglected. The semantics of the parallel state component $\mathsf{And}\ (S_1, S_2)$ is formally defined in the following:

$$\begin{aligned} &st[\![\mathsf{And}\ (S_1, S_2)]\!]x = \\ &\qquad \text{let } (y_1, S_1') = st[\![S_1]\!]x; \\ &\qquad \phantom{\text{let }} (y_2, S_2') = st[\![S_2]\!]x \\ &\qquad \text{in } \ (y_1 \cup y_2, \mathsf{And}\ (S_1', S_2')). \end{aligned}$$

The data-flow diagram in Fig. 8 depicts the signal flow of a parallel component.

[3] $\exists_1$ means that there exists exactly one.
[4] We use the curried version of δ' in our figures.

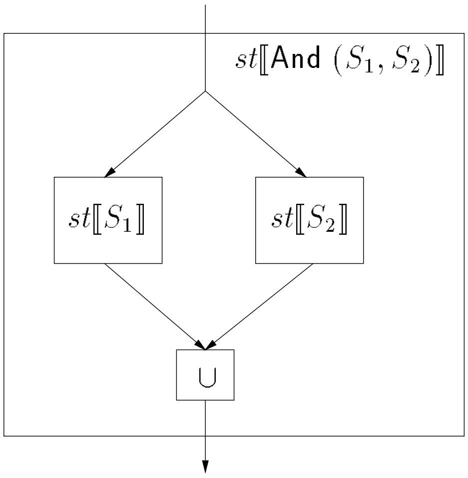

Figure 8: Parallel Composition

Local Signal-Scoping

As already mentioned, $\mathsf{Local}\ (S, L)$ for $S \in \mathcal{S}_\mathsf{D}$ and $L \in \wp_{fin}(M)$ hides any generation of $l \in L$ by S and makes S insensitive to l generated by the environment, formally denoted by (compare Fig. 9):

$$
\begin{aligned}
&st[\![\mathsf{Local}\ (S, L)]\!]x = \\
&\qquad \text{let } (y, S') = st[\![S]\!](x \backslash L) \\
&\qquad \text{in } \ (y \backslash L, \mathsf{Local}\ (S', L)).
\end{aligned}
$$

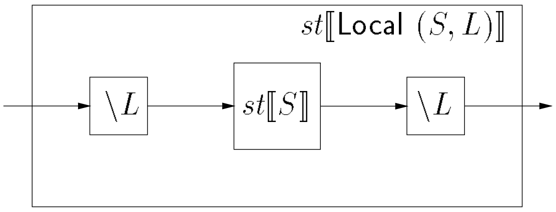

Figure 9: Local Signal-Scoping

Hierarchical Decomposition

Decomposition of a single state occurs when one wants to refine the behavior of this state. This decomposition for a sequential automaton $(\Sigma, \sigma_d, \sigma, \delta)$ is denoted by the total, finite function ϱ. An informal explanation of the following step semantics can be found in Section 2.3. As we already explained in Section 2.3, we distinguish between preemptive and non-preemptive exit of the refined state. The formal semantics of hierarchical decomposition with non-preemptive exit is denoted as follows:

$$
\begin{aligned}
&st[\![\mathsf{Dec}\ (\Sigma, \sigma_d, \sigma, \delta)\ \mathsf{by}\ \varrho]\!]x = \\
&\qquad \text{let } (\sigma', y_{master}) = \delta'(\sigma, x) \\
&\qquad \text{in } \ \text{if } \varrho(\sigma) = \mathsf{NoDec} \\
&\qquad\qquad \text{then } (y_{master}, \mathsf{Dec}\ (\Sigma, \sigma_d, \sigma', \delta)\ \mathsf{by}\ \varrho) \\
&\qquad\qquad \text{else let } (y_{slave}, S') = st[\![\pi_1(\varrho(\sigma))]\!]x \\
&\qquad\qquad\qquad \text{in } \ \text{if } ((\sigma' = \sigma) \text{ or } \pi_2(\varrho(\sigma)) = \mathsf{History})
\end{aligned}
$$

$$\begin{array}{l} \text{then } (y_{master} \cup y_{slave}, \textsf{Dec } (\Sigma, \sigma_d, \sigma', \delta) \textsf{ by } \varrho[(S', \pi_2(\varrho(\sigma)))/\sigma]) \\ \text{else } (y_{master} \cup y_{slave}, \textsf{Dec } (\Sigma, \sigma_d, \sigma', \delta) \textsf{ by } \varrho[(\mathit{init}(S'), \textsf{NoHistory})/\sigma]) \end{array}$$

where *init* is an auxiliary function which is responsible for initializing every sub-chart in $\pi_1(\varrho(\sigma))$ if history behavior is undesired:

$$\begin{array}{ll} \mathit{init}\ (\Sigma, \sigma_d, \sigma, \delta) & = (\Sigma, \sigma_d, \sigma_d, \delta) \\ \mathit{init}\ \textsf{And}\ (S_1, S_2) & = \textsf{And}\ (\mathit{init}(S_1),\ \mathit{init}(S_2)) \\ \mathit{init}\ \textsf{Local}\ (S, L) & = \textsf{Local}\ (\mathit{init}(S), L) \\ \mathit{init}\ \textsf{I-Feedback}\ (S, L) & = \textsf{I-Feedback}\ (\mathit{init}(S), L) \\ \mathit{init}\ \textsf{M-Feedback}\ (S, L) & = \textsf{M-Feedback}\ (\mathit{init}(S), L) \\ \mathit{init}\ \textsf{Dec}\ (\Sigma, \sigma_d, \sigma, \delta)\ \textsf{by}\ \varrho & = \textsf{Dec}\ (\Sigma, \sigma_d, \sigma_d, \delta)\ \textsf{by}\ \varrho' \end{array}$$

for a total, finite function ϱ' with

$$\begin{array}{l} \forall \sigma \in \Sigma : \text{if } (\varrho(\sigma) = \textsf{NoDec}) \text{ then } \varrho'(\sigma) = \textsf{NoDec} \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{else } \ \varrho'(\sigma) = (\mathit{init}\ \pi_1(\varrho(\sigma)), \pi_2(\varrho(\sigma)). \end{array}$$

For every Mini-Statechart S, *initS* provides a modified Mini-Statechart S'. All sequential automata contained in S' are reseted by the initialization procedure which means that the actual state is set to the default state.

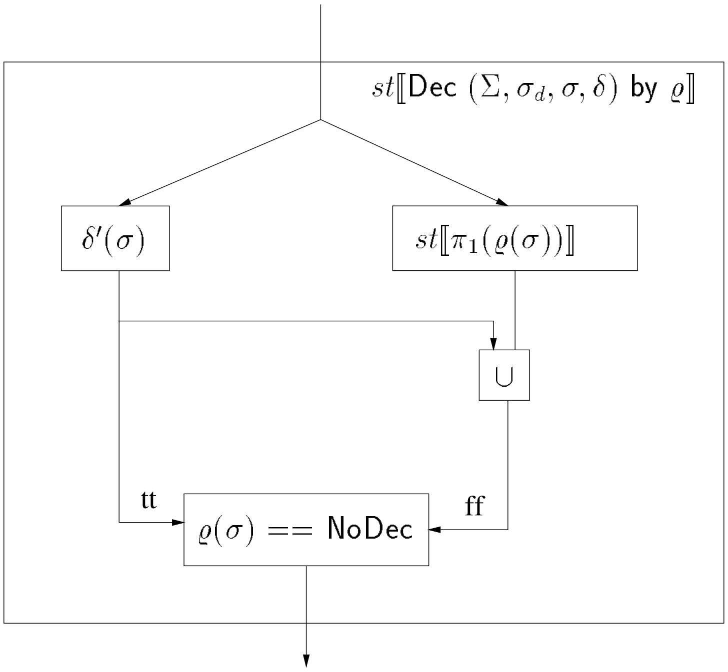

Figure 10: Hierarchical Decomposition

It is obvious why we call this kind of interrupt "non-preemptive" — whenever the master changes its state ($\sigma \neq \sigma'$), the generated signals of both master *and* slave are collected ($y_{master} \cup y_{slave}$) and the slave changes its state for the last time. If we have a history decomposed state, this change is stored in ϱ by substituting $\pi_1(\varrho(\sigma))$ by S'. Formally, this substitution is denoted by $[(S', \pi_2(\varrho(\sigma)))/\sigma]$.

The reader may wonder why we have not defined a step semantics for preemptive interrupts. Of course, we could straight forwardly slightly modify the above definition. We would get a step semantics where the output of the slave y_{slave} was neglected. If the master

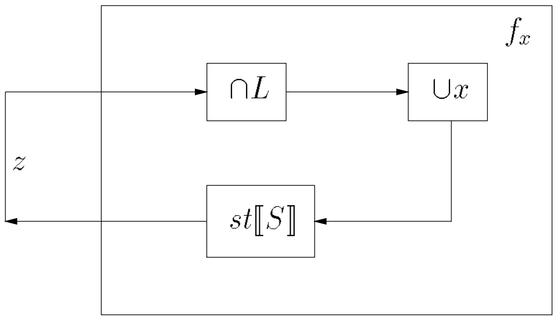

Figure 11: Instantaneous Feedback — Preparation

had been history decomposed we would not initialize the slave as in the non-preemptive case, but in contrast also would not substitute $\pi_1(\varrho(\sigma))$ by S'. However, using preemptive interrupts in combination with signal feedback a problem occurs. This problem will be explained in the next section.

3.2.2 Signal Feedback — Different Semantic Views

Semantic feedback arises out of the signal propagation mechanisms provided by Statecharts. A transition triggered by a signal set x_1 may generate signals x_2 as an action, i.e., if x_1 occurs, the transition is taken and x_2 occurs as next output set. A subset x_2' of this set may now trigger other transitions. Again, signals x_3 may be generated as action which could trigger further transitions and so on. Obviously, we get a chain reaction. In the sequel, we formalize this explanation and present two different semantic views for the feedback operator.

Instantaneous Feedback

The synchrony hypothesis [Ber89] demands that action and the event causing this action occur at the same instant of time. As a consequence, the above mentioned chain reaction instantaneously takes place. Fig. 11 depicts the situation.

The signals in z generated by Mini-Statechart S are intersected with the signals L to be fed back and then unified with the external signals in x. This signal set is passed to S at the same instant of time. Hence, to define the semantics of $st[\![\mathsf{I\text{-}Feedback}\ (S, L)]\!]x$ we have to find a solution for the following equation:

$$z = \pi_1(st[\![S]\!](x \cup (z \cap L))).$$

This can be achieved by computing a fixed point for the subsequent function:

$$\lambda z.\pi_1(st[\![S]\!](x \cup (z \cap L))).$$

We abbreviate this function by f_x. This computation may lead to the following two kinds of problems.

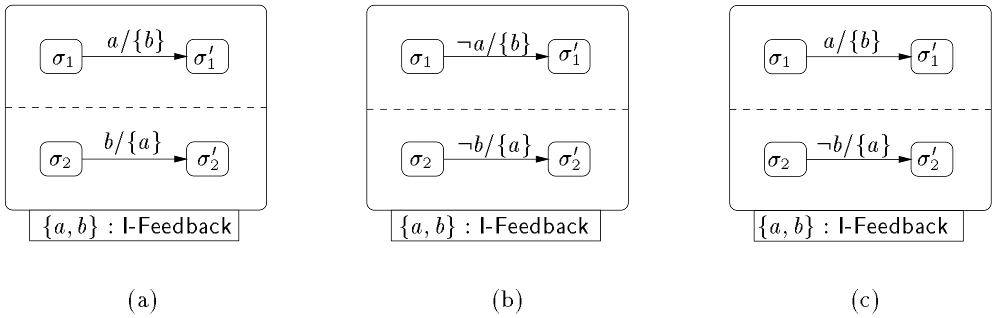

Figure 12: Problems with Fixed Points

1. Existence of Several Fixed Points

Let us consider two sequential automata

$$\begin{aligned} S_1 &= (\{\sigma_1, \sigma_1'\}, \sigma_1, \sigma_1, \delta_1) \\ S_2 &= (\{\sigma_2, \sigma_2'\}, \sigma_2, \sigma_2, \delta_2) \end{aligned}$$

where $\delta_1(\sigma_1, a) = (\sigma_1', \{b\})$ and $\delta_2(\sigma_2, b) = (\sigma_2', \{a\})$. Furthermore, let $S = \mathsf{And}\ (S_1, S_2)$, $S' = \mathsf{I\text{-}Feedback}\ (S, L)$ where $L = \{a, b\}$ (see Fig. 12(a)). Tab. 1 shows the input/output behavior of f_x.

z	$f_{\{\}}(z)$	$f_{\{a\}}(z)$	$f_{\{b\}}(z)$	$f_{\{a,b\}}(z)$
$\{\}$	$\underline{\{\}}$	$\{b\}$	$\{a\}$	$\{a,b\}$
$\{a\}$	$\{b\}$	$\{b\}$	$\{a,b\}$	$\{a,b\}$
$\{b\}$	$\{a\}$	$\{a,b\}$	$\{a\}$	$\{a,b\}$
$\{a,b\}$	$\underline{\{a,b\}}$	$\underline{\{a,b\}}$	$\underline{\{a,b\}}$	$\underline{\{a,b\}}$

Table 1: Example: Several Fixed Points

Obviously, for $f_{\{\}}$ two fixed points, namely $\{\}$ and $\{a, b\}$ exist. For this reason, [Mar92] would reject this Statechart as not being well-formed. Our approach differs in this point: we choose the least one of them, with respect to the subset ordering $\subseteq$ on signal sets. This choice seems to be reasonable, because it coincides with the result we get when operationally computing the fixed point starting with the empty stimuli set. Moreover, Statecharts should not be able to generate signals without stimulus. This property is called *causality* and is fundamental in the framework of reactive systems.

It is, however, possible to have several fixed points without having a least one. To demonstrate this, we modify δ_1 and δ_2 of the above example in the following way: $\delta_1(\sigma_1, \neg a) = (\sigma_1', \{b\})$ and $\delta_2(\sigma_2, \neg b) = (\sigma_2', \{a\})$. Fig. 12(b) shows the resulting Mini-Statechart and Tab. 2 the input/output behavior of f_x. This Statechart has both $\{a\}$ and $\{b\}$ as fixed points for the empty input set. These fixed points are not comparable with respect to the subset ordering $\subseteq$. Furthermore, $\{\}$ is not a fixed point. Hence, a least fixed point does not exist.

The reason for this result is that $\{\} \subseteq \{b\}$ but $f_{\{\}}(\{\}) \not\subseteq f_{\{\}}(\{b\})$, i.e., $f_{\{\}}$ is not monotonic with respect to the subset ordering $\subseteq$ on signal sets. An equivalent observation

z	$f_{\{\}}(z)$	$f_{\{a\}}(z)$	$f_{\{b\}}(z)$	$f_{\{a,b\}}(z)$
$\{\}$	$\{a,b\}$	$\{a\}$	$\{b\}$	$\underline{\{\}}$
$\{a\}$	$\underline{\{a\}}$	$\underline{\{a\}}$	$\{\}$	$\{\}$
$\{b\}$	$\underline{\{b\}}$	$\{\}$	$\underline{\{b\}}$	$\{\}$
$\{a,b\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$

Table 2: Example: No Least Fixed Point

can be made for $f_{\{a\}}$ and $f_{\{b\}}$. Generally, f_x may be non-monotonic if a signal occurs "negatively" in a trigger condition. However, negative signals are necessary to get deterministic automata. Whenever f_x is non-monotonic, a least fixed point possibly does not exist. This may lead to an undefined semantics. Therefore, we have to reject this kind of Mini-Statecharts. They can be detected by static analysis.

2. Nonexistence of Fixed Points

A further problem is the nonexistence of fixed points. We continue the example pictured in Fig. 12(a) by slightly modifying δ_2 to $\delta_2(\sigma_2, \neg b) = (\sigma_2', \{a\})$ (Fig. 12(c)). Again, the input/output behavior of f_x is shown in Tab. 3.

z	$f_{\{\}}(z)$	$f_{\{a\}}(z)$	$f_{\{b\}}(z)$	$f_{\{a,b\}}(z)$
$\{\}$	$\{a\}$	$\{a,b\}$	$\underline{\{\}}$	$\{b\}$
$\{a\}$	$\{a,b\}$	$\{a,b\}$	$\{b\}$	$\{b\}$
$\{b\}$	$\{\}$	$\underline{\{b\}}$	$\{\}$	$\underline{\{b\}}$
$\{a,b\}$	$\{b\}$	$\{b\}$	$\{b\}$	$\{b\}$

Table 3: Example: Nonexistence of Fixed Point

Note that $f_{\{\}}$ does not have any fixed point. Whenever a fixed point for any possible set of input signals x for a Mini-Statechart does not exist, its semantics is also undefined. Again, this is caused by the non-monotonic behaviour of f_x and we reject it as not being well-formed. Formally, the semantics of the instantaneous feedback (see Figs. 11 and 13) is defined by:

$$
\begin{aligned}
& st[\![\mathsf{I\text{-}Feedback}\ (S, L)]\!]x = \\
& \qquad \text{let } f_x = \lambda z.\pi_1(st[\![S]\!](x \cup (z \cap L))); \\
& \qquad \quad (y, S') = st[\![S]\!](x \cup (\mathit{lfp}(f_x) \cap L)) \\
& \qquad \text{in } (y, \mathsf{I\text{-}Feedback}\ (S', L)).
\end{aligned}
$$

lfp computes the least fixed point of a monotonic function and is defined as follows:

$$\mathit{lfp} : (\wp_{fin}(M) \to \wp_{fin}(M)) \to \wp_{fin}(M)$$

where $\mathit{lfp}(f_x) = \mathit{ilfp}(f_x, \{\})$ and

$$\mathit{ilfp}(f_x, M) = \text{if } f_x(M) = M \text{ then } M \text{ else } \mathit{ilfp}(f_x, f_x(M)).$$

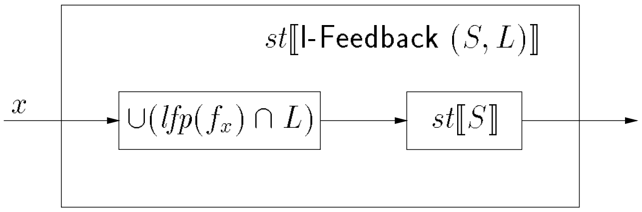

Figure 13: Instantaneous Feedback

If *lfp* is applied to a monotonic function $f_x : \wp_{fin}(M) \rightarrow \wp_{fin}(M)$ over the finite type $\wp_{fin}(M)$ then *lfp* obviously terminates. The application $lfp(f_x)$ either immediately terminates or generates a subset of a strictly higher cardinality. $\wp_{fin}(M)$ only contains subsets with finite cardinality. Let $n \in \mathbb{N}$ be an upper bound for the cardinalities of the sets in $\wp_{fin}(M)$ then n is an upper bound for the cardinalities of the sets generated by f_x, too. Thus, the function *lfp* does terminate after at most $n + 1$ steps [CE81].

The instantaneous feedback operator provides a possibility to simulate inter-level transitions. This is done by *self termination.* Self termination is carried out by inserting both an additional auxiliary state and an additional auxiliary transition. The self termination variant of Fig. 3 is pictured in Fig. 14. Instead of using one inter-level transition we utilize one internal plus one external transition (relative to C). If signal a occurs and chart C is in state A, it reacts on this signal. Simultaneously, the signal b is generated. The transition between C and B is also labeled with a b which is instantaneously fed back. Therefore, C is left after having changed its internal state from A to B. Mathematically, we have

$$\begin{array}{rcl} f_{\{a\}}(\{\}) & = & \{b\} \\ f_{\{a\}}(\{a\}) & = & \{b\} \\ f_{\{a\}}(\{b\}) & = & \{b\} \\ f_{\{a\}}(\{a,b\}) & = & \{b\} \end{array}$$

and get $lfp(f_{\{a\}}) = \{b\}$ as unique fixed point. This kind of interrupt was called non-preemptive interrupt in the last section. Remember that we did not formally define a step semantics for preemptive interrupts. The reason for this proceeding can be easily explained with self termination. For the moment, let us assume that we would have defined a preemptive version of the hierarchical decomposition. Preemptive interrupt means that the signal b would not be generated. Thus, the outermost transition could never react on b with only a as external signal, and self termination would not be possible. A fixed point does not exist. Therefore, we would have to reject this chart. Hence, we cannot achieve self-termination with preemptive interrupts.

Macro-/Micro-Step Feedback

In this section we describe a further semantic view of the feedback operator. The basis of the macro-/micro-step feedback M-Feedback (S, L) is to distinguish between signals which are generated by the environment, or *stimuli* in short, and *internal* signals which are generated by the system S itself.

We assume that a reactive system gets a set of stimuli and starts reacting on it while the

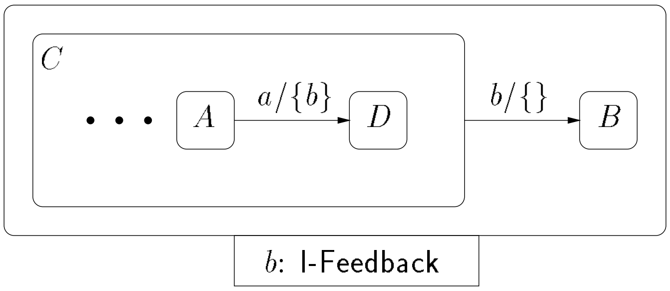

Figure 14: Self Termination

stream of external stimuli is interrupted. Internal signals are fed back, the system reacts on these signals, and proceeds until "useful" signals cannot be produced any longer. However, in contrast to the instantaneous feedback the generated signals are fed back at the next instant of time. Hence, the feedback mechanism results in a stream of signal sets. If this stream contains no "useful" signals anymore we say that the feedback operator *terminates* (see below). If the feedback terminates, the generated signals are transmitted to the environment and the next stimulus set is reacted on. Every single step of this chain reaction is called a *micro-step*, whereas a series of micro-steps, starting with the first step after the input stream was interrupted and ending with the last step before the feedback operator terminates, is called a *micro-cycle* or *macro-step*.

Different Views of the Micro-Cycle

In this section we present different views of one micro-cycle. We distinguish between lifetime of stimuli and internal signals. Lifetime of both kinds of signals can be one micro-step as well as one micro-cycle. Thus, in the sequel, four micro-step functions representing these views are defined:

$$\mu steps_n : (\mathcal{S}_\mathsf{D} \times \wp_{fin}(M)) \to \wp_{fin}(M) \to (\wp_{fin}(M) \times \mathcal{S}_\mathsf{D})^\omega$$
$$\mu steps_m : (\mathcal{S}_\mathsf{D} \times \wp_{fin}(M) \times \wp_{fin}(M)) \to \wp_{fin}(M) \to (\wp_{fin}(M) \times \mathcal{S}_\mathsf{D})^\omega$$

for $n \in \{1,4\}$ and $m \in \{2,3\}$, respectively. In the sequel, let $(y, S') = st[\![S]\!]x$ for some set of signals $x \in \wp_{fin}(M)$. Now, we distinguish four different cases.

1. Lifetime of both stimuli and internal signals is one micro-step:

$$\mu steps_1(S, L)x = (y, S')\&\mu steps_1(S', L)(y \cap L).$$

2. Lifetime of stimuli is one micro-step and lifetime of internal signals is the whole micro-cycle:

$$\mu steps_2(S, L, K)x = (y, S')\&\mu steps_2(S', L, K \cup (y \cap L))(K \cup (y \cap L)).$$

3. Lifetime of stimuli is the whole micro-cycle and lifetime of internal signals is one micro-step:

$$\mu steps_3(S, L, K)x = (y, S')\&\mu steps_3(S', L, K)(K \cup (y \cap L)).$$

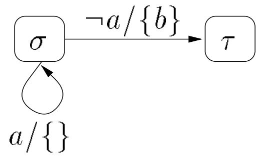

Figure 15: Restart after "Termination"

4. Lifetime of both stimuli and internal signals is the whole micro-cycle:

$$\mu steps_4(S,L)x = (y,S')\&\mu steps_4(S',L)(x \cup (y \cap L)).$$

To define the step semantics of the macro-/micro-step feedback operator, we have to discuss the notion of termination first. According to [HPSS87] a macro-step terminates if no transition is possible anymore. At first glance, this notion of termination seems to be sensible. At second glance, however, the following two problems arise. First of all, reactive systems never terminate in a classical sense. This is assured by our total transition function δ. To achieve a similar behavior as proposed in [HPSS87] we could define a macro-step to terminate if no actual state is changed and no signals are generated. However, this solution is not adequate. Because of the existence of negative trigger conditions the Mini-Statechart is able to restart if no signals are generated.

Example 6 *The automaton A in Fig. 15 shows an example for this phenomenon. Let us assume that state σ has been reached. Now, let signal a be sent by the environment. Thus, A generates the empty set of signals and stays in state σ, i.e., the actual state does not change and no signals are generated. However, the empty set of signals triggers the condition $\neg a$. Hence, A "restarts" and produces $\{b\}$ as new output.*

As a consequence, we have to define another notion of termination:

Termination of a Micro-Cycle

Let $S \in \mathcal{S}_{\mathsf{D}}$ and $L \in \wp_{fin}(M)$, then we say that $\mathsf{M\text{-}Feedback}\ (S,L)$ *terminates for stimulus* $x \in \wp_{fin}(M)$ *relative to* $f \in \{\mu steps_n(S,L) : n = 1,4\} \cup \{\mu steps_2(S,L,\{\})\} \cup \{\mu steps_3(S,L,x)\}$ *in step* $i \in \mathbb{N}$ iff there exists a set of signals $y \in \wp_{fin}(M)$ such that:

$$\forall j \in \mathbb{N} : j \geq i \Rightarrow \pi_1(f(x)\downarrow j) = y \wedge (\pi_2(f(x)\downarrow j) = \pi_2(f(x)\downarrow (j+1))).$$

This means that — beginning with step i — the feedback operation produces the same signal set y in every single successor-step and the corresponding Mini-Statechart does not change its internal structure forever. We say it has reached a *stable* state. In the sequel we will abbreviate this termination predicate to *term* (S,L,f,i,x).

The Step Semantics of the Macro-/Micro-Step Feedback

The behavior of the step semantics according to the different views mentioned above is now formally denoted by:

1. $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\mu steps_1(S,L)x)\downarrow k$ if *term* $(S,L,\mu steps_1(S,L),k,x)$ and $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\perp,\perp)$ else.

2. $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\mu steps_2(S,L,\{\})x)\downarrow k$ if $term\ (S,L,\mu steps_2(S,L,\{\}),k,x)$ and $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\perp,\perp)$ else.

3. $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\mu steps_3(S,L,x)x)\downarrow k$ if $term\ (S,L,\mu steps_3(S,L,x),k,x)$ and $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\perp,\perp)$ else.

4. $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\mu steps_4(S,L)x)\downarrow k$ if $term\ (S,L,\mu steps_4(S,L),k,x)$ and $st[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x = (\perp,\perp)$ else.

Note that only internal signals of the very last micro-step are transmitted to the environment. But it would not be hard to redefine this step semantics in such a way that all internal signals are collected and transmitted to the environment after termination. This is left to the reader. Theoretically, we only require the semidecideability of the predicate *term*. Of course the termination of each macro-step is in practice even (fully) decidable by static analysis because our Mini-Statecharts only deal with a finite state and signal space. Hence, we also could have defined a total step semantics as for the instantaneous feedback. If the micro cycle does not terminate we assign $(\perp,\perp)$ as semantics which coincides with the effect of testing termination of the micro cycle. Testing may not terminate itself which would yield a $\perp$ result.

3.3 The Stream Semantics

As mentioned in the introduction, the semantic model associates with every Mini-Statechart a stream processing function with type

$$\mathsf{D}[\![.]\!] : \mathcal{S}_{\mathsf{D}} \to \wp_{fin}(M)^{\omega} \to \wp_{fin}(M)^{\omega}.$$

The input as well as the output of our semantics is a stream of signal sets. In contrast to the step semantics, the stream semantics does not yield a subsequent Mini-Statechart. More precisely, the stream semantics of Mini-Statecharts is the iteration of the step semantics:

$$\begin{aligned}&\mathsf{D}[\![S]\!]x\&xs = \\ &\qquad \text{let } (y,S') = st[\![S]\!]x \\ &\qquad \text{in } \ y\&\mathsf{D}[\![S']\!]xs.\end{aligned}$$

The stream semantics models the complete input/output history of a Mini-Statechart in a functional way. Hence, it can be used immediately to prototype Mini-Statecharts without any further modification.

3.4 Adding the Delayed Feedback Operator

Causality problems which can occur when dealing with instantaneous feedback can be avoided by the aid of a delayed feedback. The difference between the instantaneous and the delayed feedback is that every action of an event is considered to occur at the next instant of time as additional input. The delayed feedback is more intuitive than the instantaneous and therefore suitable for the layman. However, a stepwise semantics for this operator cannot be specified because we need access to the "next" step.

In the last section we demonstrated how to derive a stream semantics from a step semantics. This straight forward development was possible because we did not allow the use of the delayed feedback operator. If we additionally integrate the delayed feedback operator, we cannot design a step semantics. The reason is that we have not only to think about "what is in a step" [PS91], but what is in a step and the very next step, too.

Thus, we have to go without a step semantics. We immediately start with the stream semantics which now contains all syntactic categories of $\mathcal{S}$. The functionality of the strict stream semantics is:

$$[\![.]\!] : \mathcal{S} \to \wp_{fin}(M)^{\omega} \to (\wp_{fin}(M) \times \mathcal{S})^{\omega}.$$

In contrast to the interpretation function $\mathsf{D}[\![.]\!]$ each output stream element not only consists of a set of signals but in addition of the subsequent Mini-Statechart. For $\mathsf{D}[\![.]\!]$ this was the task of the step function. In the sequel, we define the stream semantics for each element of the syntactic category. Note that we also have to redefine the stream semantics for *all* syntactic categories in $\mathcal{S}_{\mathsf{D}}$ because we cannot utilize the step semantics any longer by reason of its mutually recursive definition.

Sequential Automaton

$$\begin{array}{l}
[\![(\Sigma, \sigma_d, \sigma, \delta)]\!] x \& xs = \\
\qquad \text{let } (\sigma', y) = \delta'(\sigma, x) \\
\qquad \text{in } \ (y, (\Sigma, \sigma_d, \sigma', \delta)) \& [\![(\Sigma, \sigma_d, \sigma', \delta)]\!] xs.
\end{array}$$

Parallel Composition

$$\begin{array}{l}
[\![\mathsf{And}\ (S_1, S_2)]\!] x \& xs = \\
\qquad \text{let } (y_1, S_1') = ft([\![S_1]\!] x \& xs); \\
\qquad \qquad (y_2, S_2') = ft([\![S_2]\!] x \& xs) \\
\qquad \text{in } \ (y_1 \cup y_2, \mathsf{And}\ (S_1', S_2')) \& [\![\mathsf{And}\ (S_1', S_2')]\!] xs.
\end{array}$$

Local Signal-Scoping

$$\begin{array}{l}
[\![\mathsf{Local}\ (S, L)]\!] x \& xs = \\
\qquad \text{let } (y, S') = ft([\![S]\!](x \backslash L) \& xs) \\
\qquad \text{in } \ (y \backslash L, \mathsf{Local}\ (S', L)) \& [\![\mathsf{Local}\ (S', L)]\!] xs.
\end{array}$$

Hierarchical Decomposition

$$\begin{array}{l}
[\![\mathsf{Dec}\ (\Sigma, \sigma_d, \sigma, \delta)\ \mathsf{by}\ \varrho]\!] x \& xs = \\
\qquad \text{let } (\sigma', y_{master}) = \delta'(\sigma, x) \\
\qquad \text{in } \ \text{if } \varrho(\sigma) = \mathsf{NoDec} \\
\qquad \qquad \text{then } (y_{master}, \mathsf{Dec}\ (\Sigma, \sigma_d, \sigma', \delta)\ \mathsf{by}\ \varrho) \& [\![\mathsf{Dec}\ (\Sigma, \sigma_d, \sigma', \delta)\ \mathsf{by}\ \varrho]\!] xs \\
\qquad \qquad \text{else let } (y_{slave}, S') = ft([\![\pi_1(\varrho(\sigma))]\!] x \& xs) \\
\qquad \qquad \qquad \text{in } \ \text{if } ((\sigma' = \sigma) \text{ or } \pi_2(\varrho(\sigma)) = \mathsf{History}) \\
\qquad \qquad \qquad \qquad \text{then let } S'' = \mathsf{Dec}\ (\Sigma, \sigma_d, \sigma', \delta)\ \mathsf{by}\ \varrho[(S', \pi_2(\varrho(\sigma)))/\sigma] \\
\qquad \qquad \qquad \qquad \qquad \text{in } (y_{master} \cup y_{slave}, S'') \& [\![S'']\!] xs \\
\qquad \qquad \qquad \qquad \text{else let } S'' = \mathsf{Dec}\ (\Sigma, \sigma_d, \sigma', \delta)\ \mathsf{by}\ \varrho[(init(S'), \mathsf{NoHistory})/\sigma] \\
\qquad \qquad \qquad \qquad \qquad \text{in } (y_{master} \cup y_{slave}, S'') \& [\![S'']\!] xs.
\end{array}$$

Delayed Feedback

$$\begin{array}{l} [\![\mathsf{D\text{-}Feedback}\ (S,L)]\!]x_1\&x_2\&xs = \\ \quad \text{let } (y,S') = ft([\![S]\!]x_1\&x_2\&xs) \\ \quad \text{in } (y, \mathsf{D\text{-}Feedback}\ (S',L))\&[\![\mathsf{D\text{-}Feedback}\ (S',L)]\!](x_2 \cup (y \cap L))\&xs. \end{array}$$

Instantaneous Feedback

$$\begin{array}{l} [\![\mathsf{I\text{-}Feedback}\ (S,L)]\!]x\&xs = \\ \quad \text{let } f_x = \lambda z.\pi_1(ft([\![S]\!](x \cup (z \cap L))\&xs)); \\ \qquad (y,S') = ft([\![S]\!](x \cup (\mathit{lfp}(f_x) \cap L))\&xs) \\ \quad \text{in } (y, \mathsf{I\text{-}Feedback}\ (S',L))\&[\![\mathsf{I\text{-}Feedback}(S',L)]\!]xs. \end{array}$$

Macro-/Micro-Step Feedback

As we showed in the last section, there is a need to distinguish four different cases of micro-step functions. These functions were defined on the lines of the step semantics for some stimulus $x \in \wp_{fin}(M)$. Now, we cannot any longer apply the step semantics. Thus, we have to redefine the micro-step functions. Fortunately, there are only some slight modifications to do. Instead of using the definition $(y,S') = st[\![S]\!]x$, we now denote (y,S') by $ft([\![S]\!]x\&xs)$ for some arbitrary stream of sets $xs \in \wp_{fin}(M)^\omega$. The rest of the definition does not change:

$$\begin{array}{rcl} \mu steps_1(S,L)x &=& (y,S')\&\mu steps_1(S',L)(y \cap L) \\ \mu steps_2(S,L,K)x &=& (y,S')\&\mu steps_2(S',L,K \cup (y \cap L))(K \cup (y \cap L)) \\ \mu steps_3(S,L,K)x &=& (y,S')\&\mu steps_3(S',L,K)(K \cup (y \cap L)) \\ \mu steps_4(S,L)x &=& (y,S')\&\mu steps_4(S',L)(x \cup (y \cap L)). \end{array}$$

Note that xs can be arbitrary chosen because we only talk about one micro-cycle, i.e., the reaction on one stimulus. Hence, there is no need at all to consider the subsequent stimuli. However, we have to utilize an infinite stream to be able to define the stream semantics which is used to denote the micro-step functions.

Having carried through this simple modification, the micro-step functions keep their functionalities. Therefore, it is not necessary to redefine the notion of termination of a micro-cycle. Altogether, the stream semantics of the four different views of the macro-/micro-step feedback is formally denoted by:

1. $[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x\&xs = ((\mu steps_1(S,L)x)\downarrow k)\&[\![\mathsf{M\text{-}Feedback}(S',L)]\!]xs$
   if $\mathit{term}\ (S,L,\mu steps_1(S,L),k,x)$ and
   $[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x\&xs = \perp$ else, where S' denotes $\pi_2(\mu steps_1(S,L)x)\downarrow k$.
2. $[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x\&xs = ((\mu steps_2(S,L,\{\})x)\downarrow k)\&[\![\mathsf{M\text{-}Feedback}(S',L)]\!]xs$
   if $\mathit{term}\ (S,L,\mu steps_2(S,L,\{\}),k,x)$ and
   $[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x\&xs = \perp$ else where S' denotes $\pi_2(\mu steps_2(S,L,\{\})x)\downarrow k$.
3. $[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x\&xs = ((\mu steps_3(S,L,x)x)\downarrow k)\&[\![\mathsf{M\text{-}Feedback}(S',L)]\!]xs$
   if $\mathit{term}\ (S,L,\mu steps_3(S,L,x),k,x)$ and
   $[\![\mathsf{M\text{-}Feedback}\ (S,L)]\!]x\&xs = \perp$ else where S' denotes $\pi_2(\mu steps_3(S,L,x)x)\downarrow k$.

4. $[\![\textsf{M-Feedback}\ (S, L)]\!] x \& xs = ((\mu steps_4(S, L)x) \downarrow k) \& [\![\textsf{M-Feedback}(S', L)]\!] xs$
   if $term\ (S, L, \mu steps_4(S, L), k, x)$ and
   $[\![\textsf{M-Feedback}\ (S, L)]\!] x \& xs = \perp$ else where S' denotes $\pi_2(\mu steps_4(S, L)x) \downarrow k$.

3.5 Comparing the Feedback Operators

We have introduced three different kinds of feedback operators. In this section we want to discuss their advantages and drawbacks. The most important feature of the instantaneous feedback operator is that action and the event causing this action occur at the same instant of time. As a consequence, messages can be passed between components without any time going by. Due to this property, timers can easily be specified by the aid of this operator. However, the usage of the instantaneous feedback operator may cause conflicts. Several or no fixed point may exist. Hence, causality problems may occur. Though such Mini-Statecharts, in principle, can be detected by static analysis the runtime may sometimes be to high. Furthermore, I-Feedback is not easy to understand for the layman. Therefore, it might not be accepted by users.

In contrast, the delayed feedback operator has an easy intuitive semantics and is therefore suitable for the layman. Mini-Statecharts that would be rejected by I-Feedback would not by D-Feedback. Indeed, we never have to reject a chart when using the delayed feedback operator alone. The reason is that we do not have to compute any fixed points. Unfortunately, we could not define a step semantics for this operator. Applying D-Feedback there always is one tick between occurrence of the event and generation of the action. Thus, signal passing takes time. This impedes the correct handling of time signals. In particular, the D-Feedback operator cannot be used for simulating "in state" events [Inc90] whereas I-Feedback can.

The macro-/micro-step feedback operator provides a possibility to distinguish between external and internal signals. This is a feature that is desired by our industrial partners. As for the delayed feedback operator causality problems cannot occur. However, we have to concern ourselves with termination problems. This feedback operator enables us to write Mini-Statecharts that, given a set of external signals, never terminate. Non-terminating charts only have an artificial semantics ($\perp$). In practice, such charts may cause serious problems because they do not react to external signals anymore. Theoretically, non-terminating charts can be detected by static analysis. But, again, in practice, the complexity will often be to high. Further on, using a cascade of this operator implies an hierarchical notion of time. This may lead to a confusing and impenetrable semantics.

Naturally, it is possible to use all feedback operators together in one specification. However, mixing all operators may lead to a semantics that is by no means intuitively clear. In our opinion, I-Feedback provides a clean theoretical concept. Determining which feedback operator is the most appropriate one is left to future work and is hoped to be found out by the aid of case studies with our industrial partner BMW.

4 Conclusion and Future Work

We presented an abstract syntax and a compositional, denotational semantics, based on stream processing functions for so-called Mini-Statecharts, a special subclass of Statecharts. Different operators for the semantic feedback of signals were presented. The approach outlined in this paper has, compared with related work, the advantage that it introduces a very simple stepwise semantics which is easy to understand and which can easily be lifted to a stream semantics. When additionally dealing with a delayed feedback operator, it is necessary to go without a step semantics and to redefine the stream semantics. This modifications could be developed in a straight forward fashion and therefore are also easy to understand. At present, we follow up another, quite different semantic approach for Mini-Statecharts, too, where the syntax is translated into an intermediate syntactic description of a transition system based on Boolean terms.

References

[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 Verison. Technical Report TUM-I9202-2, Technische Universität München, Fakultät für Informatik, 80290 München, Germany, 1993.

[Ber89] G. Berry. Real time programming: special purpose or general purpose languages. *Information Processing 89*, 1989.

[Bro93] M. Broy. *Informatik — Eine grundlegende Einführung (Teil 2)*. Springer, Berlin/Heidelberg, 1993.

[CE81] E.M. Clark and E.A. Emerson. Characterizing Properties of Parallel Programs as Fixpoints. In *Seventh International Colloqium on Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*. Springer, 1981.

[Fre90] P. Freyd. Recursive types reduce to inductive types. In *Proceedings of the fifth annual IEEE symposium on Logic in Computer Science*. IEEE Computer Society Press, 1990.

[GS95] R. Grosu and K. Stølen. A denotational model for mobile point-to-point dataflow networks. Technical Report SFB 342/14/95 A, Technische Universität München, Fakultät für Informatik, 80290 München, Germany, 1995.

[Gun92] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. The MIT Press, 1992.

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

[Har90] D. Harel. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16:403 – 413, 1990.

[HdR91] C. Huizing and W.-P. de Roever. Introduction to design choices in the semantics of statecharts. *Information Processing Letters*, 37, 1991.

[HL95] M.P.E. Heimdahl and N.G. Leveson. Completeness and consistency analysis of state-based requirements. Proceedings on the 17th international conference on Software Engineering, pages 3 – 14. IEEE Computer Society Press, 1995.

[HPSS87] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the formal semantics of statecharts. Proceedings on the Symposium on Logic in Computer Science, pages 54 – 64, 1987.

[HRdR92] J.J.M. Hooman, S. Ramesh, and W.P. de Roever. A compositional axiomatization of statecharts. *Theoretical Computer Science*, 101:289 – 335, 1992.

[Inc90] i-Logix Inc. *Languages of Statemate.* i-Logix Inc., 22 Third Avenue, Burlington, Mass. 01803, U.S.A., January 1990.

[Mar92] F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. volume 630 of *Lecture Notes in Computer Science*, pages 550 – 564. Springer-Verlag, 1992.

[Pau87] L.C. Paulson. *Logic and Computation, Interactive Proof with Cambridge LCF*, volume 2 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1987.

[Pau92] L.C. Paulson. *ML for the Working Programmer.* Cambridge University Press, 1992.

[Pau94] L.C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science.* Springer, 1994.

[PS91] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In T. Ito and A.R. Meyer, editors, *Proccedings of the "Theoretical Aspects in Computer Software 91"*, volume 526 of *Lecture Notes in Computer Science*, pages 244 – 264. Springer-Verlag, 1991.

[SS95] B. Schätz and K. Spies. Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik. Technical Report TUM-I9529, Technische Universität München, Fakultät für Informatik, 80290 München, Germany, 1995.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* The MIT Press, 1977.

[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 : Third International Symposium Organized Jointly with the Working Group Provably Correct Systems - ProCoS*, volume 863 of *Lecture Notes in Computer Science.* Springer, 1994.