T U M

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

An Extended Version of Mini-Statecharts

Peter Scholz

TUM-I9628
Juni 1996

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

TUM-INFO-06-1996-I9628-350/1.-FI

Typescript: ---

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

An Extended Version of Mini-Statecharts *

Peter Scholz

Technische Universität München, Institut für Informatik
D-80290 München, Germany
E–Mail: 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 a visual specification mechanism for specifying reactive, embedded systems. They are implemented in commercial tools like Statemate. However, some syntactic constructs impede the modular system specification and have a confusing semantics. In [NRS96] we presented Mini-Statecharts, a lean version of Statecharts. Mini-Statecharts are restricted to the most important syntactic elements of Statecharts but are nevertheless powerful enough to specify complex systems. In this contribution, we extend the core language with local variables and integer-valued signals to avoid state explosion. We show that the formal semantics, presented in [NRS96], smoothly carries over to the semantics of the extended language.

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 allow 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 their syntax and semantics. An overview of these problems can be found in [vdB94].

In this paper, we describe a small and slender version of Statecharts, called *Mini-Statecharts*. In contrast to traditional Statecharts [Har87], Mini-Statecharts can be clearly decomposed into subcharts. Thus, they can be developed in a fully modular way by simply sticking them together. Mini-Statecharts are restricted to the most essential constructs. The basic components are sequential, deterministic automata. Mini-Statecharts can be orthogonally composed and hierarchically decomposed. We introduce three different syntactic constructs for broadcasting, which differ in their timing. A scoping mechanism to restrict broadcasting to certain subcharts is presented.

[Mar92] and [HRdR92] already provided steps in the right direction. Our work extends their approaches by local variables, integer-valued signals, and the concept of explicit feedback operators for communication. Our language has a formal and at the same time understandable semantics. It has been developed by analyzing case studies from our industrial partners.

Although Mini-Statecharts are powerful enough to describe large and complex reactive systems, we assign a concise, formal semantics to them. It is given in a fully functional way, based on the specification methodology FOCUS. Therefore, we can mix pure functional FOCUS specifications [BDD$^+$93, SS95, GS95] with Mini-Statecharts. The main intention of this paper is to demonstrate

- how to restrict and modify the syntax of traditional Statecharts [Har87] in order to get a modular specification language,
- that in contrast to related approaches we are able to define a formal, denotational, compositional semantics for Mini-Statecharts, and
- that Mini-Statecharts are not a toy language but can be used to specify practical systems with many complex states.

Furthermore, the semantics 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 the published Statecharts semantics. Even the authors of Statemate admit that the Statemate's simulation and dynamic tests tools, and its various code generators have a slightly different semantics [HN95]. In our approach there exists exactly one semantics. It can be used to prototype and simulate reactive systems as well as to reason about systems in a suitable theorem prover, like *Isabelle* [Pau94]. In the context of

verification, the availability of a compositional semantics is desirable to get manageable proofs.

We presented our core language in [NRS96]. The interested reader is referred to this report. However, we want to mention that it is not necessary to study it before reading this contribution. We here repeat the most important issues. Those readers who are interested in an formal treatment and the semantic problems that can occur, are nevertheless invited to a detailed lecture. We show that the formal semantics, presented in [NRS96], smoothly carries over to the semantics of the extended language.

This paper is structured as follows. In Section 2 we introduce the core language of Mini-Statecharts and present a concise, abstract syntax for it. For the reader who is familiar with [NRS96], most of this part is a repetition. In Section 3 we extend the core language by the concept of local variables and integer-valued signals and develop a formal semantics for it.

2 The Core Language of Mini-Statecharts

Our formalism assumes a global, discrete time. We assume that every Mini-Statechart can make a step — at least an idle step — at every single time point. This assures time progress because every single transition takes place in one time unit [GS95]. Informally speaking, every Mini-Statechart consumes and yields a sequence of sets of signals. Each element of the sequence denotes the set of signals that are present at one time unit. All other signals that are not contained in this set are assumed to be absent. Subsequent sets denote subsequent instants of time. Signals that occur between two consecutive time ticks are considered to arrive simultaneously.

In this section we propose an abstract, inductively defined textual syntax for Mini-Statecharts $\mathcal{S}$. It consists of sequential automata, parallel composition, feedback, hierarchical decomposition, and hiding. For a detailed introduction in the core language of Mini-Statecharts the interested reader is referred to [NRS96]. Let M denote a (potentially infinite) set of signal names, *States* a nonempty (potentially infinite) set of state-names, and $\mathcal{B}(M)$ the Boolean terms over M. $\wp_{fin}(X)$ denotes the set of finite subsets of some set X.

2.1 Sequential Automata

Sequential automata are the basic elements of Mini-Statecharts. The deterministic, sequential automaton

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

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

1. $\Sigma \in \wp_{fin}(\mathit{States})$ denotes the nonempty finite set of all states of the automaton.
2. $\sigma_d, \sigma \in \Sigma$ represent the default state and the current state, respectively. We need the state σ_d to initialize Mini-Statecharts for re-entering non-history, hierarchically decomposed states (see Section 3.5.4).

3. $\delta : \Sigma \times \mathcal{B}(M) \to \Sigma \times \wp_{fin}(M)$ is the finite, partial, deterministic 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, a and b must not together be present to enable the condition. Trigger conditions formulated over Boolean terms allow any combination of absent or present signals as guard.

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.

At every instant of time, A consumes a set of signals x and instantaneously produces a set of signals y, if there exists a transition with trigger condition t such that t is enabled by x and $\delta(\sigma, t) = (\sigma', y)$. Otherwise it performs an idle step, which does not have to be explicitly specified in δ. For instance, $\neg(a \wedge b)$ is enabled by the signal sets $\{\}$, $\{a\}$, and $\{b\}$ but not by $\{a, b\}$. In Section 3.5.1, we derive an equivalent, *total* state transition function δ' from δ, which is directly triggered by sets of signals instead of Boolean terms. For convenience, δ' is applied in the semantics and δ in the syntax.

2.2 Parallel Composition

Suppose S_1 and S_2 are Mini-Statecharts. Then their parallel composition is denoted by

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

This leads to a Mini-Statechart that behaves like S_1 and S_2 simultaneously: output signal sets of S_1 and S_2 are simply unified at every single time tick. In the graphical 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 the next section.

2.3 Broadcast Communication

Parallel composition is used to denote orthogonal components. However, parallel systems often are not completely independent. Therefore, Statecharts provide a broadcast communication mechanism to pass messages between components working in parallel. In [Har87] this behavior is already 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 for 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

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

are also in $\mathcal{S}$. They denote instantaneous, delayed, and macro-/microstep 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 (perfect synchrony hypothesis [BG88]) and at the next instant of time, respectively. M-Feedback distinguishes between two levels of time, namely macro- and microtime.

Example 1 (TV Set) *We introduce our syntax by the aid of an example which is adapted from [HdR91]. It models a television set with two sound levels (MUTE and SOUNDON). Only two channels (CH1 and CH2) can be received. The graphical notation is borrowed from [Har87]. The current state of every sequential automaton is characterized by a filled box and every transition between states σ and σ' is labeled with "t/x", iff $\delta(\sigma, t) = (\sigma', x)$. The feedback operator is pictured in Fig. 1 as an extra box, sticked to the bottom of the Mini-Statechart.*

When we change from one channel to another, usually the sound is turned off for a moment to avoid unwanted noise. To model this, we define two parallel components $S_{CHANNELS}$ and S_{SM} (SM for switching mode). Pressing a channel button "1", "2" on the remote control, the internal signal "sm" is generated and the TV simultaneously switches to the corresponding channel. The signal "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 "sm" and simultaneously generates "mute". The signal "mute" is also fed back and therefore S_{SOUND} reacts on "mute". Finally, the sound will be turned off. After one time tick, the signal "sound" is generated to turn it on again.*

$$\begin{aligned}
&\mathsf{I\text{-}Feedback}\ (\mathsf{And}\ (S_{CHANNELS}, \mathsf{And}\ (S_{SM}, S_{SOUND})), \{\mathrm{sm}, \mathrm{sound}, \mathrm{mute}\}) \\
&\quad S_{CHANNELS} = (\{\mathrm{CH1}, \mathrm{CH2}\}, \mathrm{CH1}, \mathrm{CH1}, \delta_{CHANNELS}) \\
&\qquad \delta_{CHANNELS}(\mathrm{CH1}, 1) = (\mathrm{CH1}, \{\mathrm{sm}\}) \\
&\qquad \delta_{CHANNELS}(\mathrm{CH1}, 2) = (\mathrm{CH2}, \{\mathrm{sm}\}) \\
&\qquad \delta_{CHANNELS}(\mathrm{CH2}, 1) = (\mathrm{CH1}, \{\mathrm{sm}\}) \\
&\qquad \delta_{CHANNELS}(\mathrm{CH2}, 2) = (\mathrm{CH2}, \{\mathrm{sm}\}) \\
&\quad S_{SM} = (\{\mathrm{SILENT}, \mathrm{LOUD}\}, \mathrm{LOUD}, \mathrm{LOUD}, \delta_{SM}) \\
&\qquad \delta_{SM}(\mathrm{LOUD}, \mathrm{sm}) = (\mathrm{SILENT}, \{\mathrm{mute}\}) \\
&\qquad \delta_{SM}(\mathrm{SILENT}, \neg\mathrm{sm}) = (\mathrm{LOUD}, \{\mathrm{sound}\}) \\
&\quad S_{SOUND} = (\{\mathrm{MUTE}, \mathrm{SOUNDON}\}, \mathrm{SOUNDON}, \mathrm{SOUNDON}, \delta_{SOUND}) \\
&\qquad \delta_{SOUND}(\mathrm{MUTE}, \mathrm{sound}) = (\mathrm{SOUNDON}, \{\}) \\
&\qquad \delta_{SOUND}(\mathrm{SOUNDON}, \mathrm{mute}) = (\mathrm{MUTE}, \{\}).
\end{aligned}$$

2.4 Hierarchical Decomposition

Mini-Statecharts include a clear and effective way to express hierarchical structures. In contrast to original Statecharts [Har87], this decomposition is fully modular because we prohibit *inter-level transitions*, i.e., transitions between states of different levels of hierarchy. Suppose that $(\Sigma, \sigma_d, \sigma, \delta)$ is a sequential automaton. Then

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

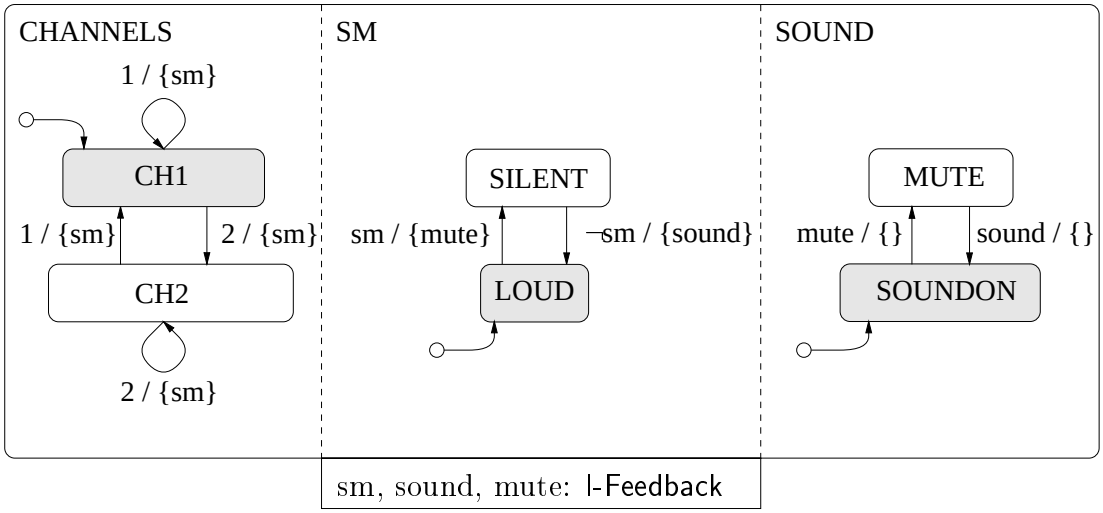

Figure 1: TV Set

is also in $\mathcal{S}$, where:

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

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 $\pi_1(\varrho(\sigma))$ 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, the slave first terminates its action concerning the current input signals and then is left. This is called non-preemptive interrupt/exit.

2.5 Hiding and Restriction

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 hiding and restriction. Suppose that S is in $\mathcal{S}$ and $L, R \in \wp_{fin}(M)$, then the constructs

$$\mathsf{Local}\ (S, L) \quad \text{and} \quad \mathsf{Restrict}\ (S, R)$$

are also in $\mathcal{S}$. $\mathsf{Local}\ (S, L)$ hides any generation of any $l \in L$ by S and makes S insensitive to any l generated by the environment. $\mathsf{Restrict}\ (S, R)$ has the opposite behavior. It restricts the input and output signals of S to signals in R. Note that these operators both are not available in conventional Statecharts. However, in our opinion they are essential to describe large reactive systems. They can be used to restrict signals to certain components of the system. The restrict operator was not yet presented in [NRS96]. Note that $\mathsf{Restrict}\ (S, R)$ can be expressed by the aid of $\mathsf{Local}\ (S, L)$ and vice versa.

3 The Extended Language of Mini-Statecharts

In spite of parallel composition and hierarchy, state explosion can occur, for example, if we extend our TV set to five channels. The result is pictured in Fig. 2. It is unthinkable to design a commercial TV set with 100 channels in this way: we would get an automaton with 100 states and 10,000 transitions.

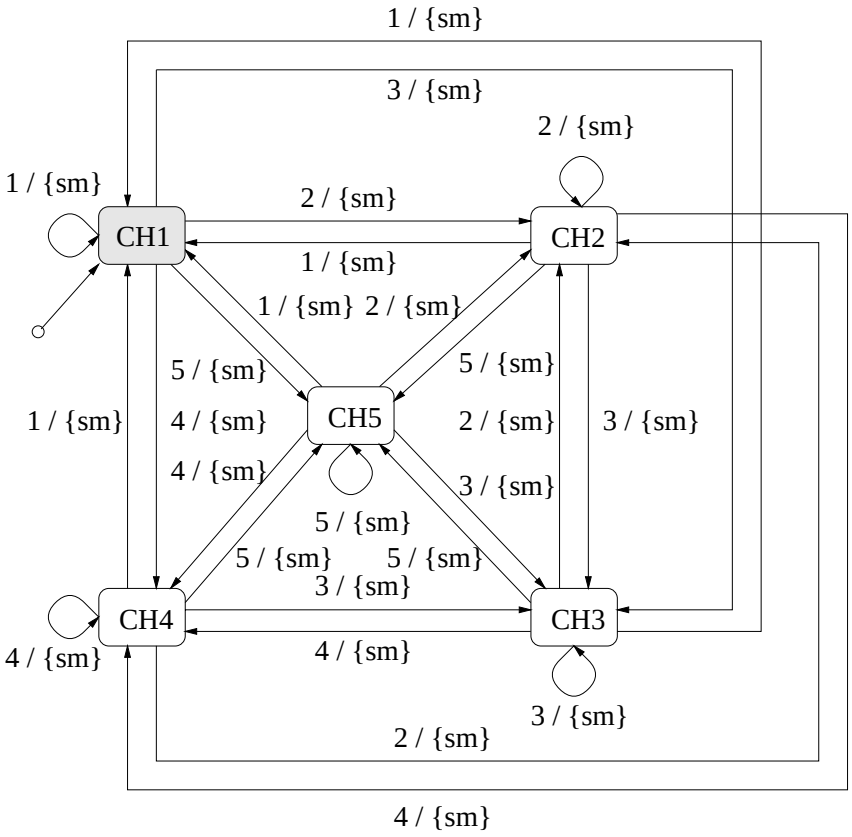

Figure 2: Example: TV

Therefore, we decided to extend Mini-Statecharts with local variables in order to avoid this state explosion. Traditional Statecharts allow to declare and access to global variables. However, global variables impede the definition of a compositional semantics. Moreover, there exist two basic concepts for communication: message passing and global variables. Traditional Statecharts incorporate both. In our opinion, there is no need to use both concepts together in one language.

In this section we propose a syntactic notation for Mini-Statecharts that has been extended by the concept of local integer variables and integer-valued signals. In contrast to a pure signal, an integer-valued signal incorporates, in addition to the information about its presence, an integer number denoting its value.

3.1 Signals, Variables, Expressions, and Commands

In contrast to Section 2, M is here disjointly partitioned in M_p and M_v, representing the set of *pure* and *integer-valued* signals, respectively. Furthermore, we assume a set V of variables. V has to be disjoint from the sets introduced so far. The other syntactic sets associated with $\mathcal{T}$, a simple language for transitions (borrowed from [Win93] and adapted for our purposes) are:

- integers Int,
- truth values $Bool = \{\mathsf{true}, \mathsf{false}\}$,
- arithmetic expressions $Aexp$,
- Boolean expressions $Bexp$, and
- commands Com.

In presenting the syntax of $\mathcal{T}$ we will follow the convention that

- n ranges over the numbers Int;
- X ranges over the variables V;
- E_v and E_p range over M_v and M_p, respectively;
- a/b range over arithmetic/Boolean expressions $Aexp/Bexp$ and
- c ranges over commands Com.

We describe the formation rules for arithmetic/Boolean expressions and commands by:

- $a ::= n \mid X \mid E_v \mid a_1 \ \mathsf{add}\ a_2 \mid a_1 \ \mathsf{sub}\ a_2 \mid a_1 \ \mathsf{mul}\ a_2$,
- $b ::= \mathsf{true} \mid \mathsf{false} \mid a_1 \ \mathsf{equ}\ a_2 \mid a_1 \ \mathsf{leq}\ a_2 \mid \ \mathsf{not}\ b \mid b_1 \ \mathsf{and}\ b_2$,
- $c ::= \mathsf{skip} \mid X := a \mid E_v := a \mid E_p \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{fi} \mid c_1; c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{od}$.

Note that we use $\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{fi}$ as an abbreviation for $\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else\ skip\ fi}$. The meaning of these expressions and commands is straightforward. In contrast to [Inc90, HN95], we use the semicolon as sequential and not as parallel composition. To see the difference, we take a look at the following example. Suppose that the command c of a transition is defined as $X := X + 1; Y := X$ and that $X = 2$ is the value X had before executing this command. Executing c in our setting would yield $Y = 3$. In [Inc90, HN95] however we would get $Y = 2$. Thus, the semicolon there signifies more "do this too" than "and then do".

However, when two or more commands want to change the same variable in the same step so-called racing conditions [HN95] can occur, which have to be detected by Statemate's simulation and dynamic test tools because the values of the variables are unknown before

runtime. In our opinion, this is complicated and superfluous. As a consequence, we have chosen the sequential execution order to get a non-ambiguous meaning and to avoid dynamic analysis.

To define the denotational semantics of $\mathcal{T}$ we first need a partial function $\gamma : M_v \to \mathbb{Z}$ that holds the value for present integer-valued signals. Here, we often interpret γ as a set G in $\wp_{fin}(M_v \times \mathbb{Z})$, where

$$\begin{aligned}\forall (m,n) \in \wp_{fin}(M_v \times \mathbb{Z}) : (m,n) \in G \Leftrightarrow \gamma(m) = n\\ \forall (m_1,n_1),(m_2,n_2) \in G : m_1 = m_2 \Rightarrow n_1 = n_2.\end{aligned}$$

We abbreviate $M_v \to \mathbb{Z}$ to Γ. We then define an environment ε as a total function $\varepsilon : V \to \mathbb{Z}$. This function also is often interpreted as a set E in $\wp_{fin}(V \times \mathbb{Z})$, where the following condition have to be fulfilled:

$$\begin{aligned}\forall (v,n) \in \wp_{fin}(V \times \mathbb{Z}) : (m,n) \in E \Leftrightarrow \varepsilon(v) = n\\ \forall (v_1,n_1),(v_2,n_2) \in E : v_1 = v_2 \Rightarrow n_1 = n_2\\ \forall v \in V\, \exists n \in \mathbb{Z} : (v,n) \in E.\end{aligned}$$

The set of all environments is denoted by $\mathcal{E}$. Note that $\mathcal{E}$ contains total functions whereas Γ only contains partial functions: variables have a defined value at every single time point, whereas signals have only when they are present. With this background we are able to define the semantic functions:

$$\begin{aligned}\mathcal{A}[\![.]\!] &: Aexp \to \Gamma \to \mathcal{E} \to \mathbb{Z}\\ \mathcal{B}[\![.]\!] &: Bexp \to \Gamma \to \mathcal{E} \to \mathbb{B}\\ \mathcal{C}[\![.]\!] &: Com \to \wp_{fin}(M) \times \Gamma \times \mathcal{E} \to \wp_{fin}(M) \times \Gamma \times \mathcal{E}.\end{aligned}$$

where $\mathbb{B} = \{tt, ff\}$. We define the denotation of an arithmetic expression, by structural induction, using the typed λ-calculus:

$$\begin{aligned}\mathcal{A}[\![n]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.n^{\mathbb{Z}}\\ \mathcal{A}[\![X]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.\varepsilon(X)\\ \mathcal{A}[\![E_v]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.\gamma(E_v)\\ \mathcal{A}[\![a_1 \text{ add } a_2]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.(\mathcal{A}[\![a_1]\!]\gamma\varepsilon + \mathcal{A}[\![a_2]\!]\gamma\varepsilon)\\ \mathcal{A}[\![a_1 \text{ sub } a_2]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.(\mathcal{A}[\![a_1]\!]\gamma\varepsilon - \mathcal{A}[\![a_2]\!]\gamma\varepsilon)\\ \mathcal{A}[\![a_1 \text{ mul } a_2]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.(\mathcal{A}[\![a_1]\!]\gamma\varepsilon * \mathcal{A}[\![a_2]\!]\gamma\varepsilon).\end{aligned}$$

Remember that every value-carrying signal E_v that occurs in a command on a transition has also to occur positively in the trigger condition. This implies that $\gamma(E_v)$ is defined. Therefore, $\mathcal{A}[\![E_v]\!]\gamma$ is also defined. The denotation of a Boolean expression is also defined by structural induction:

$$\begin{aligned}\mathcal{B}[\![\text{true}]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.tt\\ \mathcal{B}[\![\text{false}]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.ff\\ \mathcal{B}[\![a_1 \text{ equ } a_2]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.(\mathcal{A}[\![a_1]\!]\gamma\varepsilon = \mathcal{A}[\![a_2]\!]\gamma\varepsilon)\\ \mathcal{B}[\![a_1 \text{ leq } a_2]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.(\mathcal{A}[\![a_1]\!]\gamma\varepsilon \leq \mathcal{A}[\![a_2]\!]\gamma\varepsilon)\\ \mathcal{B}[\![\text{ not } b]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.\neg(\mathcal{B}[\![b]\!]\gamma\varepsilon)\\ \mathcal{B}[\![b_1 \text{ and } b_2]\!]\gamma &= \lambda\varepsilon \in \mathcal{E}.(\mathcal{B}[\![b_1]\!]\gamma\varepsilon \wedge \mathcal{B}[\![b_2]\!]\gamma\varepsilon).\end{aligned}$$

Let "let $w = g$ in f" be an abbreviation for $(\lambda w.f)g$. The definition of $\mathcal{C}[\![c]\!]$ for commands c is a bit more subtle than the definitions of $\mathcal{A}[\![.]\!]$ and $\mathcal{B}[\![.]\!]$:

$$\begin{aligned}
\mathcal{C}[\![\mathsf{skip}]\!] &= \lambda(x,\gamma,\varepsilon) \in \wp_{fin}(M) \times \Gamma \times \mathcal{E}.(x,\gamma,\varepsilon) \\
\mathcal{C}[\![X := a]\!] &= \lambda(x,\gamma,\varepsilon) \in \wp_{fin}(M) \times \Gamma \times \mathcal{E}. \\
&\quad \text{let } n = \mathcal{A}[\![a]\!]\gamma\varepsilon \text{ in } (x,\gamma,\varepsilon[n/X]) \\
\mathcal{C}[\![E_v := a]\!] &= \lambda(x,\gamma,\varepsilon) \in \wp_{fin}(M) \times \Gamma \times \mathcal{E}. \\
&\quad \text{let } n = \mathcal{A}[\![a]\!]\gamma\varepsilon \text{ in } (x \cup \{E_v\},\gamma[n/E_v],\varepsilon) \\
\mathcal{C}[\![E_p]\!] &= \lambda(x,\gamma,\varepsilon) \in \wp_{fin}(M) \times \Gamma \times \mathcal{E}.(x \cup \{E_p\},\gamma,\varepsilon) \\
\mathcal{C}[\![c_1; c_2]\!] &= \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!] \\
\mathcal{C}[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{otherwise}\ c_2\ \mathsf{fi}]\!](x,\gamma,\varepsilon) &= \begin{cases} \mathcal{C}[\![c_1]\!](x,\gamma,\varepsilon) & \text{if } \mathcal{B}[\![b]\!]\gamma\varepsilon = tt \\ \mathcal{C}[\![c_2]\!](x,\gamma,\varepsilon) & \text{else} \end{cases} \\
\mathcal{C}[\![w]\!] &= \mathcal{C}[\![\mathsf{if}\ b\ \mathsf{then}\ c; w\ \mathsf{fi}]\!]
\end{aligned}$$

where $\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{od}$ is abbreviated to w. But this involves w on both sides of the equation. For the solution of this kind of recursive equations we refer to [Win93]. We write $\gamma[n/E_v]$ for the function obtained from γ by replacing its value in E_v by n.

The execution of commands, separated by the semicolon is strictly sequential. For example, $E_v := 1; X := E_v + 1; E_v := E_v + 2$ yields $X = 2$ and $E_v = 3$. This means that even though the value of E_v in the current step is 1 and in the next step 3, E_v can change its value between these two time points. However, to get a well-defined semantics, the value that is used for communication is $E_v = 3$.

3.2 Extended Sequential Automata

Applying the concepts introduced above, we have to modify the syntactic notation for our sequential automata and get:

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

where Σ, σ_d and σ are as in Section 2. The following, additional syntactic constraints must hold:

1. $V_l \in \wp_{fin}(V)$ denotes the set of local, i.e., private read/write variables. These variables can be only read and/or written by the automaton itself. They have to be initialized:

2. $\beta_d : V_l \to Int$ is a finite, total function that describes the initial values of the local variables.

Furthermore, δ has to be modified:

$$\delta : \Sigma \times \mathcal{B}(M) \to \Sigma \times Com$$

is the finite, partial state transition function that takes a state and a Boolean term and yields the subsequent state together with a command, describing the modification of the

internal variables and the generation of pure or value-carrying signals. In contrast to the version of δ that was used in our core language, here $\wp_{fin}(M)$ is substituted by Com. This means that in the extended language, an action does not only consist of the generation of a set of (pure) signals, but of a whole command.

There is a further syntactic restriction on δ. For every transition with label t/c the following must be valid: each integer-valued signal E_v that occurs on the right-hand-side of an assignment in c also has to occur either "before" on the left-hand-side of another assignment in c or positively in t. This condition must be fulfilled in order to guarantee a defined value for E_v. In this context, to *occur positively* means that we must be able to derive that E_v is present in the trigger condition t. This is the case whenever $t \Rightarrow E_v$ is a tautology. As in the core language we assume that every transition, i.e., every command can be computed in exactly one instant of time.

Note that the trigger condition is a Boolean term as in the core language. Also for integer-valued signals, we only check absence or presence but not their values. As a consequence, the determinism of even the extended automata is easily decidable by static analysis and we can avoid dynamic analysis.

In the following we want to demonstrate how a TV with 100 channels can be specified using this kind of deterministic automaton (Fig. 3). In addition, state-of-the-art TV sets provide the opportunity to simply switch through the programs by incrementing or decrementing the channel number. This can be done with buttons "$\oplus$" and "$\ominus$", modeled as pure signals. Moreover, we have one integer-valued signal "changeto". In the graphical notation every transition between states σ and σ' is now labeled with "t/c", iff $\delta(\sigma, t) = (\sigma', c)$. The partial function β_d is assumed to initialize the unique local variable of Fig. 3 X by 1.

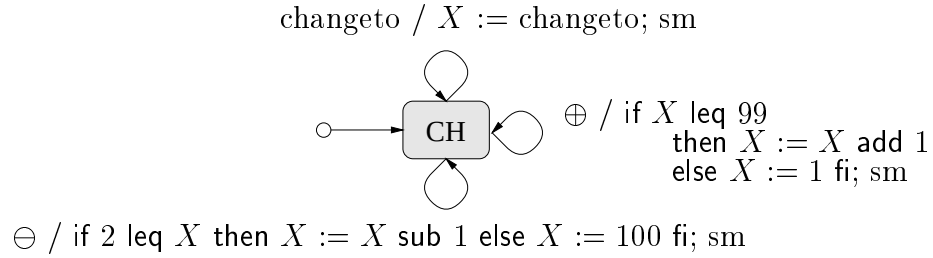

Figure 3: A TV with 100 channels, specified in the extended language

The textual version of Fig. 3 is defined in the sequel:

$$
\begin{aligned}
S = (&\{X\}, \beta_d, \{\mathrm{CH}\}, \mathrm{CH}, \mathrm{CH}, \delta) \text{ where} \\
&\delta(\mathrm{CH}, \text{changeto}) = (\mathrm{CH}, X := \text{changeto}; \text{sm}) \\
&\delta(\mathrm{CH}, \oplus) = (\mathrm{CH}, \text{if } X \leq 99 \text{ then } X := X \text{ add } 1 \text{ else } X := 1 \text{ fi}; \text{sm}) \\
&\delta(\mathrm{CH}, \ominus) = (\mathrm{CH}, \text{if } 2 \leq X \text{ then } X := X \text{ sub } 1 \text{ else } X := 100 \text{ fi}; \text{sm})
\end{aligned}
$$

and the partial function β_d is assumed to initialize the unique local variable X by 1.

3.3 Extended Hierarchical Decomposition

The usage of the extended language also enforces a redefinition of the decomposition operator. Suppose that $(V_l, \beta_d, \Sigma, \sigma_d, \sigma, \delta)$ is an extended sequential automaton. Then

$$\mathsf{Dec}\ (V_l, \beta_d, \Sigma, \sigma_d, \sigma, \delta)\ \mathsf{by}\ \varrho\ \mathsf{res}\ \varphi$$

is an extended, hierarchical decomposed Mini-Statechart, where the decomposition function ϱ is slightly adapted to the extended language:

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

is a total, finite function. Note that ϱ is modified. In addition to the possibility to choose whether a master state is history refined or not, we now can specify whether we want to initialize all local variables of the slave when reentering it or not. This is denoted by Refresh and NoRefresh, respectively. φ denotes the resolution function and is defined in the sequel.

3.4 Resolution of Conflicts

Using integer-valued signals some problems can occur. Let us assume that each of two parallel components S_1, S_2 tries to broadcast the integer-valued signal E_v. Furthermore, we suppose that S_1 assigns 21 to E_v, while at the same instant of time S_2 assigns 42. In this case, we get a semantic conflict. However, the orthogonally composed Mini-Statechart And (S_1, S_2) must produce the signal E_v with a unique value. Hence, we introduce a total *resolution* function φ, which resolves this conflict and produces a unique value:

$$\varphi : \wp_{fin}(\mathcal{S} \times M \times \mathbb{Z}) \to \mathbb{Z}.$$

For every set of conflicting integers, φ yields the integer that will be calculated when a conflict occurs. Using sets of triples of the form $(S, x, n) \in \mathcal{S} \times M \times \mathbb{Z}$ as possible input values for φ, we can define subtle resolution functions. For example, let $M = \{a, b\}$ then we can define:

- $\varphi_1(\{(S_1, a, 3), (S_2, a, 4)\}) = 3$,
  $\varphi_1(\{(S_1, b, 5), (S_2, b, 6)\}) = 5$.
  φ_1 is a resolution function that always prefers the output of chart S_1, independent of the signal name.

- $\varphi_2(\{(S_1, a, 3), (S_2, a, 4)\}) = 3$,
  $\varphi_2(\{(S_1, b, 5), (S_2, b, 6)\}) = 6$.
  In this case, φ_2 is a resolution function that prefers the output of chart S_1 whenever a conflict for signal a occurs, while for b chart S_2 is preferred.

- $\varphi_3(\{(S_1, a, 3), (S_2, a, 4)\}) = 7$,
  $\varphi_3(\{(S_1, b, 5), (S_2, b, 6)\}) = 11$.
  Here, φ_3 simply adds all conflicting values.

Of course, there are many other alternatives to define the resolution function. We include φ in the syntactic notation of the parallel composition and get instead of And (S_1, S_2):

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

In addition, contradictory integer-valued signals also can emerge when employing the communication operators. Though in this case we do not have two parallel, conflicting Mini-Statecharts, we can get conflicts between signals from the environment and signals that are fed back for communication. Thus, the operators for the delayed, instantaneous, and macro-/microstep feedback operator are also straightforwardly adapted:

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

Signal conflicts also can occur whenever applying the hierarchical decomposition. This can be the case when both master and slave broadcast the same signal with different integer values. Hence, we also have to specify a resolution function for the hierarchical decomposition and get Dec $(V_l, \beta_d, \Sigma, \sigma_d, \sigma, \delta)$ by ϱ res φ.

The remaining constructs of the core language, Local (S, L) and Restrict (S, R), need not to be modified.

3.5 The Formal Semantics of Mini-Statecharts

Reactive systems continuously interact with their environment. Thus, to define their semantics, their complete input/output behavior has to be described. This can be done by communication histories. We model the communication history of Mini-Statecharts by streams carrying tuples of sets of (pure and integer-valued) signals together with values of integer-valued signals. Mathematically, we describe the behavior of Mini-Statecharts by stream processing functions. Hence, we briefly discuss the notion of streams and stream processing functions. For a detailed description we refer to [BDD+93] and [SS95].

Given a set X of signals, a stream over X, denoted by X^ω, is an infinite sequence of elements from X. Our notation for the concatenation operator is &. Given an element x of type X and a stream s over X, the term $x\&s$ denotes the stream that starts with the element x followed by the stream s. The destructor *ft* selects the first element of a stream. A stream processing function is a function with type $X^\omega \to X^\omega$. Besides the constructor and the destructor we need an auxiliary function $s \downarrow k$ that yields for a positive natural number k the k-th element of stream s.

The definition of the semantics for the macro-/microstep feedback operator causes problems concerning the compositionality. We will give an example that reflects the situation in Section 3.5.7. In order to get a compositional semantics also for this operator, we have to introduce a new special signal.

Let † be an extra signal that is not yet contained in M. The occurrence of † in a signal set of the output stream indicates that the Mini-Statechart has not changed its current state(s) in this step. This signal will be called *stop (signal)*. It is needed to indicate the end of a so-called micro-cycle (see Section 3.5.7). In the sequel, the set $M \cup \{\dagger\}$ will be abbreviated by $M_\dagger$. Note that the set of signals $L \in \wp_{fin}(M)$, which shall be fed back,

does not contain the stop signal. The functionality of the denotational semantics is

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

This semantics is denoted as a higher order function. For its formal definition, we use an auxiliary higher order function of type

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

to take into account current/successor environment and successor chart. For $S \in \mathcal{S}$ and $s \in (\wp_{fin}(M) \times \Gamma)^\omega$, $\mathcal{D}[\![S]\!]s$ is defined by

$$\mathcal{D}[\![S]\!]s = \mathit{strip}\ ([\![S]\!](\mathit{refresh}\ S)\ s)$$

where $\mathit{strip}\ (w, x, y, z)\&s = (w, x)\&(\mathit{strip}\ s)$. The auxiliary function *refresh* initializes the environment according to the initialization functions β_d. We now define the stream semantics for all syntactic constructs of Mini-Statecharts.

3.5.1 Sequential Automata

Informally, a sequential, deterministic, and reactive automaton $(V_l, \beta_d, \Sigma, \sigma_d, \sigma, \delta)$ takes a set of (pure and value-carrying) input signals, the so-called *stimuli*, reacts on it while manipulating its own, local variables, produces a set of (pure and value-carrying) signals as output and then behaves like an automaton with modified current state and modified environment function. Note that the local variables are not visible to other automata. Thus, communication is done by events only. In contrast to basic Mini-Statecharts, value-carrying events are now allowed. If we would restrict ourself to pure signals, extended Mini-Statecharts would communicate exactly like Mini-Statecharts, presented in [NRS96].

The transition function δ is defined on Boolean terms. Reactive systems, however, have to react on 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. The function *trigger* is exactly defined as in the core language.

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

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

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

Note that *trigger* is exactly the same as in [NRS96]. To get a semantics which deals with sets of signals instead of Boolean terms, we consider in the sequel a total, deterministic state transition function δ' with the functionality

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

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

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

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[1]:

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

This property ensures δ' to be a well-defined function. Notice that the above definition of reactiveness and determinism are defined on the semantics. However, it is straight forward to formulate syntactic definitions of reactiveness and determinism:

- δ is reactive, if $\forall \sigma \in \Sigma : \left(\bigvee_{t \in T_\delta(\sigma)} t\right) \Leftrightarrow tt$.
- δ is deterministic, if $\forall \sigma \in \Sigma_A \, \forall t_1, t_2 \in T_\delta(\sigma) : t_1 \neq t_2 \Rightarrow (t_1 \wedge t_2 \Leftrightarrow ff)$.

where $T_\delta(\sigma) := \{t \in \mathcal{B}(M) \mid \exists \sigma' \in \Sigma, c \in Com : \delta(\sigma, t) = (\sigma', c)\}$.

Besides simulation, *Statemate* [Inc90], 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.

$$\begin{aligned} [\![(V_l, \beta_d, \Sigma, \sigma_d, \sigma, \delta)]\!]\varepsilon\,(x, \gamma)\&s = \text{let } & (\sigma', c) = \delta'(\sigma, x); \\ & (y, \gamma', \varepsilon') = \mathcal{C}[\![c]\!](x, \gamma, \varepsilon); \\ & S' = (V_l, \beta_d, \Sigma, \sigma', \delta) \\ \text{in } & \text{if } \sigma \neq \sigma'; \\ & \text{then } (y, \gamma', \varepsilon', S')\&([\![S']\!]\varepsilon'\, s) \\ & \text{else } (y \cup \{\dagger\}, \gamma', \varepsilon', S')\&([\![S']\!]\varepsilon'\, s). \end{aligned}$$

The sequential automaton takes the current environment ε together with the tuple (x, γ) in every time point. x represents the set of all (pure and integer-valued) signals that are currently present. γ is defined for all integer-valued signals that are contained in x. $\gamma(E_v)$ denotes the current integer value for all signals E_v in $x \cap M_v$.

[1] $\exists_1$ means that there exists exactly one.

In the case that the automaton changes its current state ($\sigma \neq \sigma'$) the semantics instantaneously yields the quadruple $(y, \gamma', \varepsilon', S')$. Here y denotes the set of generated output signals, γ', ε', and S' the successors for γ, ε, and S, respectively. If the automaton does not change its current state, this is indicated by the additional output of †. After that, the automaton behaves like the automaton with modified current state.

3.5.2 Parallel Composition

The parallel composition of And (S_1, S_2, φ) behaves as S_1 and S_2 synchronously together. Generated signals of the parallel components are unified, denoted by $y_1 \cup y_2$ and $\gamma_1 \cup_\varphi \gamma_2$, where $\cup$ denotes the standard union and $\cup_\varphi$ the union of integer-valued signals w.r.t. φ. The union of the environments ε_1 and ε_2 has to be performed with care. Both ε_1 and ε_2 are total function on V. However, unifying them must yield a total function again. This is achieved by $\varepsilon_1 \cup_{V_1}^{V_2} \varepsilon_2$, which is ε_i for all variables in V_i with $i \in \{1, 2\}$: Here, V_i denotes the set of signals that are used in chart S_i.

$$
\begin{aligned}
[\![\textsf{And}\ (S_1, S_2, \varphi)]\!]\varepsilon\,(x, \gamma)\&s = \text{let}\ & (y_1, \gamma_1, \varepsilon_1, S_1') = ft([\![S_1]\!]\varepsilon\,(x, \gamma)\&s); \\
& (y_2, \gamma_2, \varepsilon_2, S_2') = ft([\![S_2]\!]\varepsilon\,(x, \gamma)\&s); \\
& y' = y_1 \cup y_2;\ \gamma' = \gamma_1 \cup_\varphi \gamma_2;\ \varepsilon' = \varepsilon_1 \cup_{V_1}^{V_2} \varepsilon_2; \\
& S' = \textsf{And}\ (S_1', S_2', \varphi) \\
\text{in}\ & \text{if}\ \dagger \in y_1 \cap y_2 \\
& \text{then}\ (y', \gamma', \varepsilon', S')\&([\![S']\!]\varepsilon'\ s) \\
& \text{else}\ (y' \backslash \{\dagger\}, \gamma', \varepsilon', S')\&([\![S']\!]\varepsilon'\ s).
\end{aligned}
$$

And (S_1, S_2, φ) does not change its current states, if both S_1 and S_2 do not change theirs, which is indicated by $\dagger \in y_1 \cap y_2$. Note that an equivalent condition to this would be $S_1 = S_1' \wedge S_2 = S_2'$. The reader might now why we did not choose this condition instead of $\dagger \in y_1 \cap y_2$. One might argue that we then even could define your semantics without stop signal. However, we will precisely explain the reason for our strategy in Section 3.5.7.

The formal semantics of And (S_1, S_2, φ) demonstrates the advantage of our compositional semantics: to define $[\![\textsf{And}\ (S_1, S_2, \varphi)]\!]$ we just have to calculate $[\![S_1]\!]$ and $[\![S_1]\!]$ and then put the results together.

3.5.3 Hiding and Restriction

As already mentioned, Local (S, L) and Restrict (S, R) for $S \in \mathcal{S}$ and $L, R \in \wp_{fin}(M)$ are used for encapsulation, which is formally denoted by:

$$
\begin{aligned}
[\![\textsf{Local}\ (S, L)]\!]\varepsilon\,(x, \gamma)\&s = \text{let}\ & (y, \gamma', \varepsilon', S') = ft([\![S]\!]\varepsilon(x \backslash L, \gamma|_{M_v \backslash L})\&s); \\
& S'' = \textsf{Local}\ (S', L) \\
\text{in}\ & (y \backslash L, \gamma'|_{M_v \backslash L}, \varepsilon', S'')\&[\![S'']\!]\varepsilon'\ s
\end{aligned}
$$

$$
\begin{aligned}
[\![\textsf{Restrict}\ (S, R)]\!]\varepsilon\,(x, \gamma)\&s = \text{let}\ & (y, \gamma', \varepsilon', S') = ft([\![S]\!]\varepsilon(x \cap R, \gamma|_{M_v \cap R})\&s); \\
& S'' = \textsf{Restrict}\ (S', R) \\
\text{in}\ & (y \cap R, \gamma'|_{M_v \cap R}, \varepsilon', S'')\&[\![S'']\!]\varepsilon'\ s.
\end{aligned}
$$

Again, $\gamma|_{M_v \cap R}$ denotes the restriction of γ on signals in $M_v \cap R$. It is obvious that one of these constructs can be considered to be an abbreviation: either Restrict (S, R) can be defined as Local $(S, M \backslash R)$ or Local (S, L) as Restrict $(S, M \backslash R)$.

3.5.4 Hierarchical Decomposition

Decomposition of a single state occurs when one wants to refine the behavior of this state. This decomposition for a sequential automaton $(V_l, \beta_d, \Sigma, \sigma_d, \sigma, \delta)$ is denoted by the total, finite function ϱ. The formal semantics of hierarchical decomposition is denoted as follows, where $(V_l, \beta_d, \Sigma, \sigma_d, \sigma, \delta)$ is abbreviated to A. Variables with index m and s denote master and slave, respectively.

(1) $[\![\mathsf{Dec}\ A\ \mathsf{by}\ \varrho_A\ \mathsf{res}\ \varphi]\!]\varepsilon\,(x,\gamma)\&s =$
(2) let $(y_m, \gamma_m, \varepsilon_m, A') = ft([\![A]\!]\varepsilon\,(x,\gamma)\&s)$
(3) in if $\varrho(\sigma) = \mathsf{NoDec}$
(4) then let $S' = \mathsf{Dec}\ A'\ \mathsf{by}\ \varrho$
(5) in $(y_m, \gamma_m, \varepsilon_m, S')\&[\![S']\!]\varepsilon_m\ s$
(6) else let $f = \text{if } \pi_3(\varrho(\sigma)) = \mathsf{Refresh} \text{ then } (\mathit{refresh}\ \pi_1(\varrho(\sigma))) \text{ else } \mathit{id}$;
(7) $(y_s, \gamma_s, \varepsilon_s, S') = ft([\![\pi_1(\varrho(\sigma))]\!](f\,\varepsilon)\,(x,\gamma)\&s)$;
(8) $y = \text{if } \dagger \in y_m \cap y_s \text{ then } y_m \cup y_s \text{ else } (y_m \cup y_s)\backslash\{\dagger\}$;
(9) $\gamma' = \gamma_m \cup_\varphi \gamma_s$;
(10) $\varepsilon' = \varepsilon_m \cup_{V_l}^{V_s} \varepsilon_s$
(11) in if ($\dagger \in y_m$ or $\pi_2(\varrho(\sigma)) = \mathsf{History}$)
(12) then let $S'' = \mathsf{Dec}\ (\Sigma, \sigma_d, A', \delta)\ \mathsf{by}\ \varrho[(S', \pi_2(\varrho(\sigma)), \pi_3(\varrho(\sigma)))/\sigma]$
(13) in $(y, \gamma', \varepsilon', S'')\&[\![S'']\!]\varepsilon'\ s$
(14) else let $S'' = \mathsf{Dec}\ (\Sigma, \sigma_d, A', \delta)\ \mathsf{by}\ \varrho[(\mathit{init}(S'), \mathsf{NoHistory}, \pi_3(\varrho(\sigma)))/\sigma]$
(15) in $(y, \gamma', \varepsilon', S'')\&[\![S'']\!]\varepsilon'\ s$.

To define the semantics of $\mathsf{Dec}\ A\ \mathsf{by}\ \varrho_A$ we first let make the master one step, denoted by $ft([\![A]\!]\varepsilon(x,\gamma)\&s)$ in line (2). If the current state σ of the master is not decomposed at all (3) $\varrho(\sigma) = \mathsf{NoDec}$, the semantics immediately proceeds to the next step like a for a pure sequential automaton (5).

Otherwise (6), if $\varrho(\sigma) \neq \mathsf{NoDec}$, then $\pi_1(\varrho(\sigma))$ denotes the slave and we use the following abbreviations: f represents the function *refresh* if $\pi_3(\varrho(\sigma)) = \mathsf{Refresh}$, i.e., if the local variables of the slave shall be initialized and otherwise the function *id*. *id* represents the identity and leaves the current environment unchanged, whereas $(\mathit{refresh}\ \pi_1(\varrho(\sigma)))$ initializes all variables of the slave according to their default values.

Similar to the parallel composition, $\mathsf{Dec}\ A\ \mathsf{by}\ \varrho_A$ only generates a stop signal in the current step, when both master and slave generate one. γ' denotes the values of the integer-valued signals $y \cap M_v$ that are present in the next step; ε' denotes the new environment. In (10), V_l denotes the local variables of A and V_s all variables of the slave $\pi_1(\varrho(\sigma))$.

In line (11) we have again to distinguish between two different cases. If A does not change its current state ($\dagger \in y_m$), the semantic function proceeds to the next step, where the slave has to be modified. This is achieved by substituting S' for S in ϱ (12). The same must be done if the state σ of the master is history decomposed. However, if the master changes its current state from σ to σ' ($\sigma \neq \sigma'$ is indicated by $\dagger \notin y_m$) and σ is not history decomposed, then the slave must be initialized (14),(15). *init* is defined according to [NRS96]: $\mathit{init}(S')$ initializes all sequential automata in S' to their default states. Note that *init* does not initialize variables.

3.5.5 Delayed Communication

In [NRS96] we demonstrated that broadcast communication is the critical point of the language. We presented three different feedback operators. In the extended language, the delayed feedback is, like in [NRS96], also the one with the "easiest" formal semantics:

$$\begin{aligned}
&[\![\textsf{D-Feedback } (S, L, \varphi)]\!]\varepsilon\,(x_1, \gamma_1)\&(x_2, \gamma_2)\&s = \\
&\quad \text{let } (y, \gamma', \varepsilon', S') = ft([\![S]\!]\varepsilon\,(x_1, \gamma_1)\&(x_2, \gamma_2)\&s); \\
&\qquad S'' = \textsf{D-Feedback } (S', L, \varphi) \\
&\quad \text{in } (y, \gamma', \varepsilon', S'')\&([\![S'']\!]\varepsilon'\,(x_2 \cup (y \cap L), \gamma_2 \cup_\varphi \gamma'|_{M_v \cap L})\&s).
\end{aligned}$$

The tuples (x_1, γ_1) and (x_2, γ_2) denote the input signals of the current and the next instant of time, respectively. Signals (y, γ') are instantaneously generated and fed back as additional input in the next time point: $(x_2 \cup (y \cap L), \gamma_2 \cup_\varphi \gamma'|_{M_v \cap L})$, where $\gamma|_{M_v \cap L}$ denotes the restriction of γ on signals in $M_v \cap L$. Whenever a conflict for integer-valued signals occur, the resolution function φ specifies whether the environment or the component itself wins recognition.

3.5.6 Instantaneous Communication

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 delayed feedback now instantaneously takes place. 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 one step of the semantics of $\textsf{I-Feedback } (S, L, \varphi)$, i.e., $ft([\![\textsf{I-Feedback } (S, L, \varphi)]\!]\varepsilon\,(x, \gamma)\&s)$ we have to find a solution for the following equation:

$$z = \pi_1(ft([\![S]\!]\varepsilon\,(x \cup (z \cap L), \gamma)\&s)).$$

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

$$\lambda z.\pi_1(ft([\![S]\!]\varepsilon\,(x \cup (z \cap L), \gamma)\&s)).$$

We abbreviate this function by $f^{\varepsilon}_{x,\gamma}$. Because of negative trigger conditions, some problems can emerge when defining the formal semantics of this operator. This problems and how to solve them was discussed in detail in [NRS96]. We there showed that we must reject certain Mini-Statecharts, which are lacking in unique fixed points. Charts to be rejected can be detected by static analysis. In this contribution, we assume that unproper charts already have been rejected. Formally, the semantics of the instantaneous feedback for not rejected charts is defined by:

$$\begin{aligned}
&[\![\textsf{I-Feedback } (S, L, \varphi)]\!]\varepsilon\,(x, \gamma)\&s = \\
&\qquad \text{let } f^{\varepsilon}_{x,\gamma} = \lambda z.\pi_1(ft([\![S]\!]\varepsilon\,(x \cup (z \cap L), \gamma)\&s)); \\
&\qquad\quad (y, \gamma', \varepsilon', S') = ft([\![S]\!](x \cup (lfp(f^{\varepsilon}_{x,\gamma}) \cap L))\&s); \\
&\qquad\quad \gamma'' = \gamma \cup_\varphi \gamma'; \\
&\qquad \text{in } (y, \gamma'', \varepsilon', S')\&([\![\textsf{I-Feedback } (S', L, \varphi)]\!]\varepsilon'\,s)
\end{aligned}$$

where *lfp* computes the least fixed point of a monotonic function w.r.t. the subset ordering and is defined as follows:

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

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

$$\mathit{ilfp}(f_x, y) = \text{if } f^{\varepsilon}_{x,\gamma}(y)\backslash\{\dagger\} = y \text{ then } y \text{ else } \mathit{ilfp}(f^{\varepsilon}_{x,\gamma}, f^{\varepsilon}_{x,\gamma}(y)\backslash\{\dagger\}).$$

In the sequel, we want to demonstrate the functionality of *lfp*. Let us take a look at Fig. 4. We assume that the environment currently produces $x = \{a, b\}$, where the value of a and b is 1 and 2, respectively, i.e., $\gamma(a) = 1$ and $\gamma(b) = 2$. For all other signals γ is undefined in the current time point. Both automata do not have any local variables and so we simply have $\varepsilon = \emptyset$. First of all, we get $f^{\varepsilon}_{x,\gamma}(\emptyset) = \{b\}$. Applying this function once more yields $f^{\varepsilon}_{x,\gamma}(\{b\}) = \{b, c\}$. The last application produces $f^{\varepsilon}_{x,\gamma}(\{b, c\}) = \{b, c\}$ and a fixed point is reached.

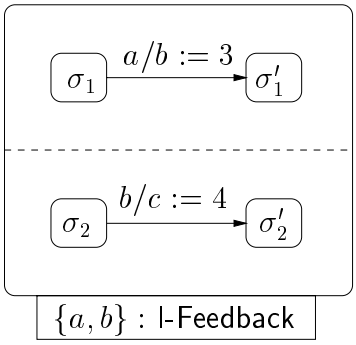

Figure 4: Example: instantaneous feedback

Due to the definition of our state transition function the concept of [NRS96] smoothly carries over to the extended language. Note that we only can achieve this result because transitions are triggered by the presence or absence of (even pure) signals. If transitions were also triggered by the values of signals, it would not be possible to lift the concept for instantaneous feedback as easy as demonstrated.

3.5.7 Macro-/Micro-Step Communication

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 x which are generated by the environment, or *stimuli* in short, and *internal* signals y which are generated by the system S itself.

We assume that a reactive system gets a set of stimuli x and starts reacting (1.1)-(1.4),(2.1) on it while the stream s of external stimuli is interrupted (2.1). Internal signals are fed back (1.4), the system reacts on these signals, and proceeds until "useful" signals cannot be produced any longer (1.2),(1.3). However, in contrast to the instantaneous feedback the generated signals are fed back at the next instant of (micro) time (1.4). Hence, the feedback mechanism results in a stream of signal sets (1.1)-(1.4) and we get different levels of system time. If this stream contains no "useful" signals anymore we say that the feedback operator *terminates* (see below) (1.3). If the feedback terminates, the generated signals are transmitted to the environment and the next stimulus set is reacted on (2.3). Every single step (1.1) of this chain reaction is called a *micro-step*, whereas a series of micro-steps (1.1)-(1.4), starting with the first step after the input stream was interrupted

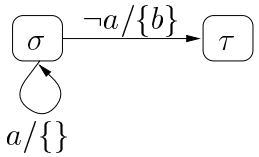

Figure 5: Restart after "Termination"

and ending with the last step before the feedback operator terminates, is called a *micro-cycle* or *macro-step*. In one macro-step we can distinguish eight different variants for lifetime of stimuli and internal signals. Lifetime of both kinds of signals can be one micro-step as well as the whole micro-cycle. However, due to space limitations in this paper we only present the following variant (for some arbitrary stream t):

$$\begin{aligned}
\mu steps(S,L)\varepsilon\,(x,\gamma) = \quad & \text{let } (y,\gamma',\varepsilon',S') = ft([\![S]\!]\varepsilon\,(x,\gamma)\&t) && (1.1)\\
& \text{in if } (\dagger \in y) \wedge (y \cap L = x) \wedge \gamma'|_{L\cap M_v} = \gamma && (1.2)\\
& \text{then } (y,\gamma',\varepsilon',S)\&(\{\dagger\},S)^{\infty} && (1.3)\\
& \text{else } (y,\gamma',\varepsilon',S')\&\mu steps(S',L)\varepsilon'\,(y\cap L,\gamma'|_{L\cap M_v}). && (1.4)
\end{aligned}$$

where lifetime of both stimuli x and internal signals y is one micro-step (1.4). For the other variants, the interested reader is referred to [NRS96]. To define the 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 current 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 2 *The automaton A in Fig. 5 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 current 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:

Let $S \in \mathcal{S}$ and $L \in \wp_{fin}(M)$ then we say that $\mu steps(S,L)\varepsilon\,(x,\gamma)$ *terminates for stimulus* $(x,\gamma) \in \wp_{fin}(M) \times \Gamma$ and (current) environment ε in step $k \in \mathbb{N}$ iff

$$k = \min\{i \in \mathbb{N} \mid \forall j > i : \{\dagger\} = \pi_1((\mu steps(S,L)\varepsilon\,x)\downarrow j)\}.$$

This means that — beginning with step k — the feedback operation produces the same signal set in every single successor-step and the corresponding Mini-Statechart does not change its internal structure forever (1.2). We say it has reached a *stable* state. In the sequel, we will abbreviate this termination predicate to $term\,(S,L,k,(x,\gamma),\varepsilon)$. The behavior of the stream semantics is now formally denoted by:

$$
\begin{aligned}
[\![\text{M-Feedback } (S,L)]\!]\varepsilon\,(x,\gamma)\&s = \quad & \text{if } \exists k \in \mathbb{N} : \mathit{term}\,(S,L,k,(x,\gamma),\varepsilon) && (2.1)\\
& \text{then let } (y,\gamma',\varepsilon',S') = (\mu \mathit{steps}(S,L)\varepsilon\,(x,\gamma))\downarrow k && (2.2)\\
& \text{in } (y,\gamma',\varepsilon',S')\&[\![\text{M-Feedback}(S',L)]\!]\varepsilon'\,s && (2.3)\\
& \text{else } \bot. && (2.4)
\end{aligned}
$$

Note that only internal signals of the very last, i.e., the k-th micro-step are transmitted to the environment (2.3). However, 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. Theoretically, we only require the semidecideability of the predicate *term* (2.1). 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 $\bot$ as semantics (2.4) which coincides with the effect of testing termination of the micro cycle. Testing may not terminate itself which would yield a $\bot$ result.

We now want to motivate the need of the stop signal. We say that our denotational semantics is compositional if for all $S_1, S_2 \in \mathcal{S}$ the following is valid [Win93]:

$$\mathcal{D}[\![S_1]\!] = \mathcal{D}[\![S_2]\!] \Longrightarrow \mathcal{D}[\![C(S_1)]\!] = \mathcal{D}[\![C(S_2)]\!].$$

This definition needs the notion of *context*. In our setting, a context $C(.)$ intuitively is a Mini-Statechart $S \in \mathcal{S}$ with exactly one "hole" (.) of type $\mathcal{S}$. In this hole we can plug another Mini-Statechart S'. Defining the semantics for the macro-/microstep feedback operator without stop signal, we can find Mini-Statecharts that produce the same output streams, but do not agree in all contexts. This will be demonstrated in Example 3.

Example 3 (Motivation for the Stop Signal) *Let S and S' denote the automata that are pictured in Fig. 6 (a) and (b), respectively. We suppose that the environment supplies both components with the input stream $\emptyset^\omega$. Once being initiated, both automata start reacting forever because all transitions are labeled with "true".*

First of all, we assume that we would have defined the semantics without stop signal, then S_1 produces an infinite stream of which all all elements are empty signal sets by proceeding idle steps. In contrast, S_2 toggles between σ' and σ'' but also produces an infinite stream of empty sets. We get $\mathcal{D}[\![S_1]\!](\emptyset,\emptyset)^\omega = \mathcal{D}[\![S_2]\!](\emptyset,\emptyset)^\omega = (\emptyset,\emptyset)^\omega$ as denotational semantics.

Let us now assume that both automata are embedded in the Macro-/Microstep Feedback operator, i.e., we have M-Feedback $(S_1,\emptyset)$ *and* M-Feedback $(S_2,\emptyset)$*. In contrast to above, now $\emptyset$ can be fed back. As we will see in the sequel, this is not trivial.* M-Feedback $(S_1,\emptyset)$ *takes the first element of the input streams, makes an idle step, produces the empty set of signals, and terminates. Now the chart can consume a further element of the input stream and starts reacting on that set again. In spite of this complicated internal behavior, like S_1, also* M-Feedback $(S_1,\emptyset)$ *produces $(\emptyset,\emptyset)^\omega$ as output.*

M-Feedback $(S_2,\emptyset)$*, in contrast, has a different behavior. Here, the micro cycle never terminates because the automaton carries on changing its current state forever. Hence, we get $\bot$ as overall semantics. Thus, the overall result is $\mathcal{D}[\![$*M-Feedback $(S_1,\emptyset)]\!] \neq \mathcal{D}[\![$*M-Feedback $(S_2,\emptyset)]\!]$ *in contradiction to the compositionality.*

The introduction of the stop signals easily solves this problem, because we get $\mathcal{D}[\![S_1]\!](\emptyset,\emptyset)^\omega = (\{\dagger\},\emptyset)^\omega \neq (\emptyset,\emptyset)^\omega = \mathcal{D}[\![S_2]\!](\emptyset,\emptyset)^\omega$ as denotational semantics and S_1, S_2 can be distinguished by their output streams.

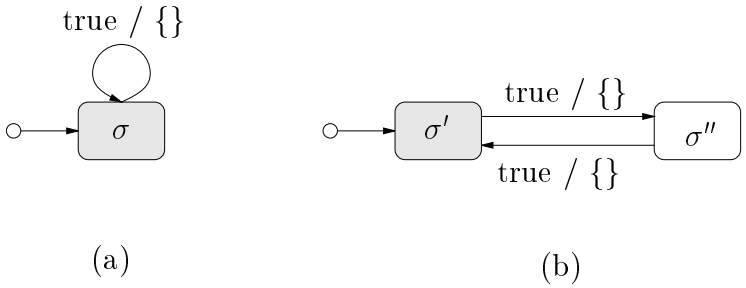

Figure 6: Motivation for the stop signal

The reason for the failure of a semantics without this extra signal is near at hand. Though without using the stop signal we could check whether the current state has changed with a condition like $S = S'$, we never would recognize this *internal* behavior in the output stream. As a consequence, we would distinguish the semantics of components that one the one hand have the same output stream, but on the other hand a different termination behavior.

4 Conclusion and Future Work

We presented the textual and visual specification language "Mini-Statecharts". Mini-Statecharts are a subclass of Statecharts, which were first introduced by David Harel. We restricted our language to the essential syntactic constructs of Statecharts. Mini-Statecharts are, in contrast to Harel's Statecharts, well-suited for the *modular* development of parallel, reactive systems. Because of their modularity, we were able to assign a compositional, formal semantics to them.

However, the author admits that the assumption of a global clock impedes the usage of Mini-Statecharts for the specification of distributed systems. Each distributed component normally is driven by its own, local clock. Thus, the communication of distributed components has to be synchronized. To develop a formal semantics that deals with a number of local clocks instead of one single, global clock is left to future work.

Also left to further work is the development of a formal semantics for a non-deterministic version of Mini-Statecharts. This goal can be achieved either by using sets of stream processing functions or relations. Non-determinism is the most appropriate possibility to express underspecification. Refining a non-deterministic specification step by step, we get a deterministic and therefore implementable specification in the end. In the semantics, refinement is denoted by set inclusion. Each refinement step must not enlarge the set of possible implementations. To formalize this iterative refinement process, we have to develop a refinement calculus. It fixes the set of feasible syntactic transformations for every refinement step.

Acknowledgment

Thanks are owed to Manfred Broy, Olaf Müller, Christian Prehofer and especially Jan Philipps who read an earlier version of this paper and provided many helpful comments.

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.

[BG88] G Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. Technical Report 842, INRIA, 1988.

[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, 1995.

[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.

[HN95] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. Submitted to: ACM Transations Software Engineering Methods, 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.

[NRS96] D. Nazareth, F. Regensburger, and P. Scholz. Mini-Statecharts: A Lean Version of Statecharts. Technical Report TUM-I9610, Technische Universität München, 1996. Also available in the WWW: http://wwwbroy.informatik.tu-muenchen.de/reports/TUM-I9610.html.

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

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

[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.

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